НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації і управління

До захисту допущено:
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім'я, прізвище)

"___"_____2020 р.

Дипломний проєкт
на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: « Інтелектуальна система моделювання популяцій тварин та аналізу природного відбору»

Виконав: студент *IV* курсу, групи *ІС-63*

Аршулік Наталія Олексіївна _____
(прізвище, ім'я, по батькові) (підпис)

Керівник *доц., к.ф.-м.н., доц. Рибачук Людмила Віталіївна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Консультант з графічної документації *доц., к.т.н., доц. Тєлишева Тамара Олексіївна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент *ст.викл, к.т.н., Солдатова Марія Олександрівна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Пояснювальна записка
до дипломного проєкту

на тему: Інтелектуальна система моделювання популяцій тварин та аналізу природного відбору

Київ – 2020 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проєкту складається з 6 розділів, містить 15 рисунків, 8 таблиць, 1 додаток, 5 джерел.

Дипломний проєкт присвячений розробці застосування для наукової та освітньої сфер у галузі популяційної біології. Застосування включає комплекс задач для підвищення ефективності роботи науковців.

Розділ загальних положень включає в себе опис предметного середовища, огляд наявних аналогів та постановку задачі. Також цей розділ включає опис діяльності функціональних складових системи.

У розділі інформаційного забезпечення визначені вхідні та вихідні дані, структура їх збереження та використані формати даних.

Розділ математичного забезпечення присвячений аналізу даних, отриманих у результаті роботи симуляції.

Програмне забезпечення включає діаграми послідовності, класів, компонентів, специфікацію функцій. Також представлено опис архітектури системи, програмного та апаратного забезпечення.

У технологічному розділі наведено керівництво користувача, опис випробувань застосування.

КЛЮЧОВІ СЛОВА: ПОПУЛЯЦІЯ, ПРИРОДНІЙ ВІДБІР, UNITY-3D, СИМУЛЯЦІЯ, АНАЛІЗ ПОПУЛЯЦІЇ.

ABSTRACT

Structure and scope of work. The explanatory note of the diploma project consists of 6 sections containing 15 figures, 8 tables, 1 supplement, 5 sources.

The diploma project is devoted to the development of applications for scientific and educational spheres in the field of population biology. The application includes a set of tasks to increase the efficiency of scientists.

The section of general provisions includes a description of the subject environment, an overview of existing analogues and problem statement. Also, this section includes a description of the functional components of the system.

The section of information support defines input and output data, the structure of their storage and used data formats.

The section of mathematical support is devoted to the analysis of the data received as a result of work of simulation.

The software includes sequence diagrams, classes, components, function specifications. There is also a description of the system architecture, software and hardware.

The technological section provides a user manual, a description of application tests.

KEY WORDS: POPULATION, NATURAL SELECTION, UNITY-3D, SIMULATION, POPULATION ANALYSIS.

ЗМІСТ

ВСТУП **6**

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ **8**

1.1 Опис предметного середовища 8

1.1.1 Опис процесу діяльності *9*

1.1.2 Опис функціональної моделі *10*

1.2 Огляд наявних аналогів 13

1.3 Постановка задачі 15

1.3.1 Призначення розробки *15*

1.3.2 Цілі та задачі розробки *15*

Висновок до розділу 15

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ **17**

2.1 Вхідні дані 17

2.2 Вихідні дані 20

2.3 Структура масивів інформації 23

Висновок до розділу 24

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ **25**

3.1 Змістовна постановка задачі 25

3.2 Математична постановка задачі 25

3.3 Обґрунтування методу розв'язання 25

3.4 Опис методів розв'язання 26

Висновок до розділу 30

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ **31**

4.1 Засоби розробки 31

4.2 Вимоги до технічного забезпечення 32

Вимоги до технічних засобів *32*

4.2.1 Загальні вимоги *32*

4.3 Архітектура програмного забезпечення 33

4.3.1 Діаграма класів *34*

4.3.2 Діаграма послідовності *35*

4.3.3 Діаграма компонентів *37*

4.3.4 Специфікація функцій *38*

ВИСНОВОК ДО РОЗДІЛУ 41

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ 42

5.1 КЕРІВНИЦТВО КОРИСТУВАЧА 42

5.2 ВИПРОБУВАННЯ ПРОГРАМНОГО ПРОДУКТУ 47

5.2.1 Мета випробувань *47*

5.2.2 Загальні положення *47*

5.2.3 Результати випробувань *48*

ВИСНОВОК ДО РОЗДІЛУ 500

ЗАГАЛЬНІ ВИСНОВКИ 52

ПЕРЕЛІК ПОСИЛАНЬ 53

ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ 54

ВСТУП

У сучасному світі, майже кожен науковий прорив, від медичної сфери до освоєння космосу, є прямим наслідком розвитку інформаційних технологій. Як головний рушій прогресу, сфера ІТ повинна приділяти увагу не тільки комерційним проєктам, а розвивати області науки далекі від бізнесу.

На сьогоднішній день екологічні проблеми набувають глобальних масштабів. Зниження глобального біорізноманіття та зростання числа видів, яким загрожує зникнення, призвели до необхідності глибше вивчити механізми стійкості і динаміки популяцій, а також і розробити методи для прогнозування ризиків вимирання. Для прогнозування необхідні дані, проте спостереження за популяціями та аналіз отриманих результатів потребують значних ресурсів та часу. Для полегшення аналізу спостережень були розроблені математичні моделі, проте і у них є свої недоліки - низька наочність або недостатня комплексність. Також проблемою є неможливість впливу на досліджуване середовище - як приклад, неможливість керування часом або мутаціями.

Для розв’язання вищенаведеного переліку проблем у рамках даного дипломного проєкту було розроблено застосування для моделювання та аналізу популяцій тварин.

Дипломний проєкт присвячений розробці альтернативного способу проведення популяційних спостережень за допомогою побудови віртуальної моделі популяції та розробці комплексу задач по аналізу середовища. Ефективність роботи науковців, що мають змогу віртуально відтворювати та змінювати природні системи, очікувано буде вище, оскільки це дає змогу отримувати дані, які важко отримати спостереженням за реальними природними системами. Також віртуальні моделі мають функції, які неможливі у не віртуальному середовищі, що надає унікальні можливості додаткового впливу на модель. Автоматизація аналізу даних дозволяє зберегти час на побудови математичних моделей та розрахунки. Візуальне

відображення даних та можливість спостерігати за тваринами безпосередньо в симуляції, надає більшу доступність та наочність даних.

Визначення та терміни що стосуються теми дипломного проєкту

У роботі [4] визначені основні терміни предметного середовища.

Популяція — сукупність організмів одного виду, що займають обмежений ареал (територія поширення якогось об'єкта або явища) та ізольовані від інших популяцій даного виду.

Популяційна біологія — галузь біології, що досліджує популяції організмів, їх зміни та взаємодію, зокрема досліджує популяційні аспекти екології, еволюції, процесів відтворення, старіння та смерті.

Популяційна екологія або демекологія — розділ загальної екології, який займається вивченням популяцій, їхньою структурою та динамікою чисельності.

Динаміка популяцій — це процеси зміни її основних біологічних показників (чисельності, біомаси, структури) в часі в залежності від екологічних факторів.

Ареал — територія поширення деякої систематичної групи організмів.

Вимирання — часткове або цілковите зникнення окремих видів, груп рослин і тварин або цілих флор і фаун з певної території чи акваторії.

Практичне значення одержаних результатів.

Розроблено застосування для моделювання популяцій тварин та аналізу природного відбору.

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Предметним середовищем даного дипломного проєкту є віртуальна модель, що відтворює реальну екологічну нішу існування тварин.

Екологічна ніша — термін в екології, що описує взаємовідносини виду або популяції з екосистемою.

Модель — це абстрактне уявлення реальності в будь-якій формі, призначене для подання певних аспектів цієї реальності з метою отримання відповідей на досліджувані питання. Віртуальна модель - це модель, що реально не існує, проте відтворює реальність при певних умовах.

Основним призначенням спостережень за екологічними нішами є отримання інформації, яка буде використана для зупинення зниження глобального біорізноманіття. Інтелектуальна система моделювання популяцій тварин та аналізу природного відбору повинна надавати змогу користувачеві створювати симуляцію навколишнього середовища за заданими параметрами, генерувати нові дані у симуляції та проводити їх аналіз. Застосунок має надавати змогу передивлятись статистичну інформацію щодо популяції тварин створеної моделі, таку як загальна кількість тварин у популяції, щільність популяції, середні показники по обраному виду тварин, просторову структуру популяції. Також, інтелектуальна система моделювання популяцій тварин повинна проводити аналіз вікової та статевої структури популяції, динамічні характеристики (народжуваність та смертність, вірогідність виживання на одиницю часу), надавати інформацію щодо росту та структури популяції.

Побудова симуляції навколишнього середовища має надавати змогу отримати дані про популяції, використовуючи дані про загальні характеристики виду, такі як гострота органів сприйняття, швидкість, необхідна кількість їжі щодня, швидкість розмноження, тощо.

1.1.1 Опис процесу діяльності

Користувач обирає або створює мапу, яка визначає кількість водних ресурсів у симуляції та їх розміщення. Далі у застосунку є змога налаштувати стартові параметри локації, такі як кількість рослинної їжі, швидкість оновлення видів рослинних ресурсів, їх щільність (налаштування цього параметра надає змогу створити як симуляції лісу, так і симуляції пустель). Також є можливість налаштування ландшафту за допомогою системи генерації схилів та перешкод (камені, пні, тощо).

Після задання критеріїв локації, користувач має змогу створити та детально налаштувати параметри вида тварин. Перелік вхідних даних цього розділу буде наведено у розділі “Визначення вхідних та вихідних даних” даного звіту.

Крім створення видів, користувач має змогу створити бажану кількість екземплярів кожного виду, та налаштувати початкове співвідношення самців і самок у популяції.

Після запуску віртуальної моделі, користувач має змогу керувати часом у симуляції, а за допомогою керування камери спостерігати за тваринним та рослинним світом. Також користувач має змогу переглядати дані щодо створенної моделі у графічному інтерфейсі користувача та отримувати аналіз по бажаним статичним характеристикам популяції.

Дії, які користувач повинен мати змогу виконати, показані за допомогою структурної схеми діяльності, зображеної на рисунку 1.1 та наведеної у графічному додатку.

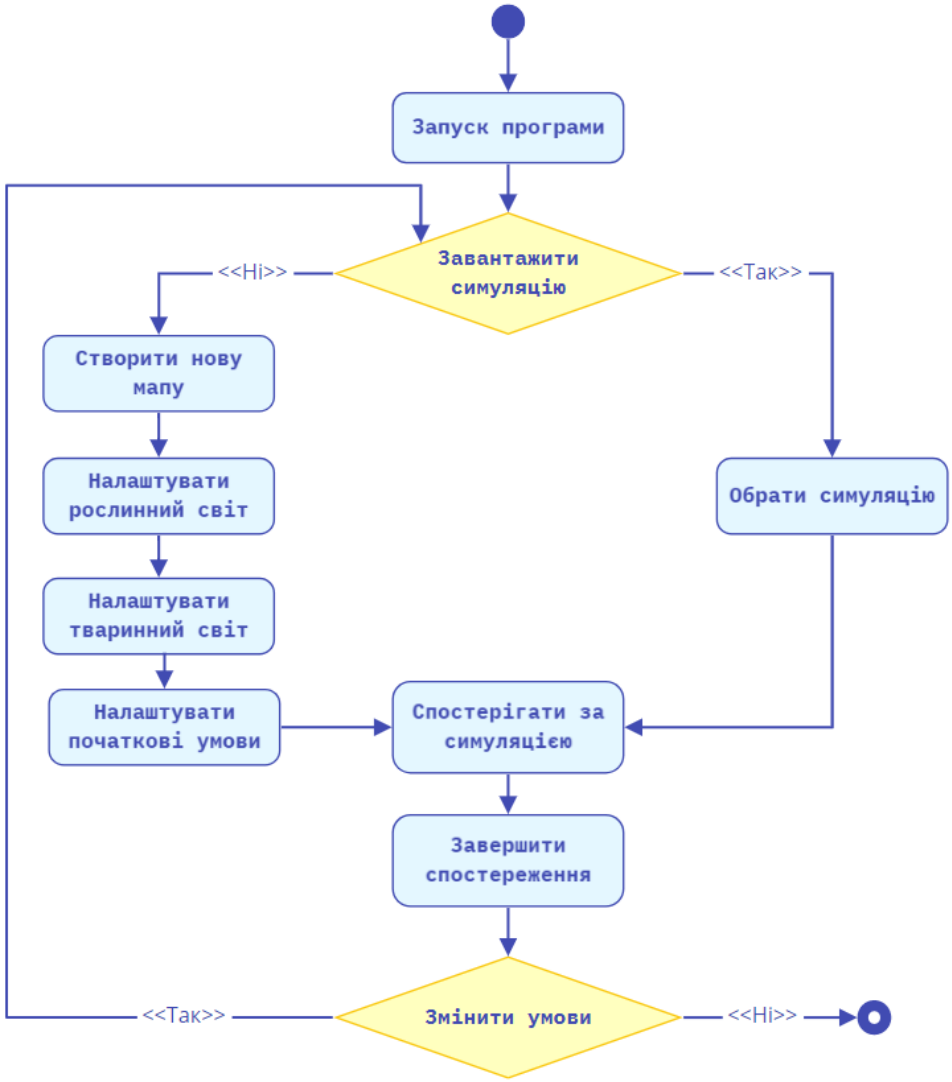

Рисунок 1.1 – Структурна схема діяльності користувача системи

1.1.2 **Опис функціональної моделі**

Для роботи системи необхідні вхідні дані для побудови бажаної віртуальної моделі та користувач, який створить віртуальне середовище з цими даними. Результатом роботи застосунка будуть нові (згенеровані у симуляції та отримані за допомогою аналіза) дані.

Визначимо дійових осіб (акторів) для подальшого проектування структурної схеми використання, а також дії у системі, які може виконувати кожен з акторів.

Актором в системі є користувач системи. Користувач займається введенням даних та моніторингом діяльності системи.

Варіанти використання:

- перегляд списку збережених симуляцій;
- завантаження існуючої симуляції;
- створення нових симуляцій;
- налаштування ландшафту нових симуляцій;
- налаштування рослинної фауни нових симуляцій;
- налаштування стартових умов нових симуляцій;
- створення нових видів;
- збереження симуляції;
- видалення симуляції;
- перегляд статистичної інформації;
- перегляд симуляції;
- керування часом у симуляції;
- збереження отриманого аналізу у файл.

Діаграма варіантів використання представлена на рисунку 1.2 та наведена у графічному додатку.

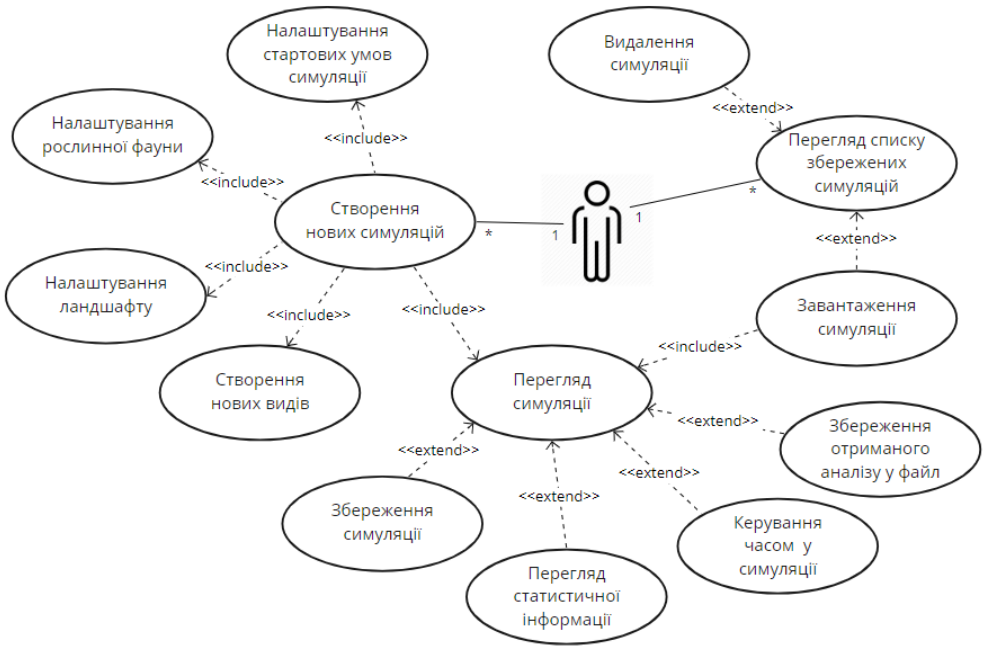

Рисунок 1.2 – Діаграма варіантів використання

Кожен з варіантів використань має власні вимоги та пріоритети.

Функціональні вимоги до варіантів використання наведені у таблиці 1.1.

Таблиця 1.1 – Вимоги до варіантів використань

Актор	Варіант використання	Опис вимог варіанта використання	Пріоритет
Користувач системи	Перегляд списку збережених симуляцій	RQ 001. Система має надавати змогу користувачу переглядати список збережених симуляцій	Середній
	Завантаження існуючої симуляції	RQ 002. Користувач повинен мати змогу переглядати список збережених симуляцій та завантажувати обрану.	Середній
	Створення нових симуляцій	RQ 003. Користувач повинен мати змогу створити нову симуляцію.	Високий
	Налаштування ландшафту нових симуляцій	RQ 004. Користувач повинен мати змогу налаштувати ландшафт нової симуляції.	Високий
	Налаштування рослинної фауни нових симуляцій	RQ 005. Система повинна надавати змогу користувачу налаштувати рослинну фауну нової симуляції.	Високий
	Налаштування стартових умов	RQ 004. Користувач повинен мати змогу налаштувати ландшафт нової симуляції.	Високий
	Створення нових видів	RQ 006. Система має надавати змогу створювати нові види тварин.	Високий
	Збереження симуляції	RQ 007. Користувач повинен мати змогу зберегти симуляцію, вказав її назву.	Середній

Кінець таблиці 1.1

Актор	Варіант використання	Опис вимог варіанта використання	Пріоритет
Користувач системи	Видалення симуляції	RQ 008. Користувач повинен мати змогу переглядати список збережених симуляцій та видаляти обрану.	Середній
	Перегляд статистичної інформації	RQ 009. Користувач повинен мати можливість відкривати і закривати вікно перегляду інформації (аналізу). RQ 010. Система має надавати змогу користувачу переглядати статистику щодо кожного виду	Високий
	Перегляд симуляції	RQ 011. Система має надавати можливість к перегляду симуляції. RQ 012. Користувач повинен мати змогу керувати камерою.	Середній
	Керування часом у симуляції	RQ 013. Користувач повинен мати змогу змінювати швидкість часу у симуляції.	Високий
	Збереження отриманого аналізу у файл	RQ 014. Користувач повинен мати змогу зберегти інформацію про початкові дані та аналіз у txt-файл.	Низький

1.2 Огляд наявних аналогів

Серед аналогів можна виділити систему Coding Adventure: Simulating an Ecosystem, розроблену Sebastian Lague, окремий функціонал якої дозволяє крім моделювання природного відбору динамічно генерувати локації за допомогою градієнтних шумів (Шум Перлина). Застосунок дозволяє дослідити вплив ландшафту та водних ресурсів на динаміку популяції. Проте,

через обмежену кількість врахованих даних, застосунок дозволяє отримати недостатньо реалістичну модель, та тільки модель типу “1 вид хижаків - 1 вид жертв”. Також у Coding Adventure: Simulating an Ecosystem відсутній інтерфейс користувача, що ускладнює аналіз стану системи.

Подібні недоліки має і веб-застосунок Deer mouse population simulation - обмежена кількість врахованих даних та можливість дослідити взаємодію лише двох видів. Крім цього, дана система має обмеження на кількість об'єктів у симуляції, що робить неможливим дослідження великих популяційних груп.

Окремо можна відзначити розробку Simulating Natural Selection авторства Primer, функціонал якої дозволяє досліджувати мутації всередині виду. Із недоліків - застосунок дозволяє досліджувати тільки один вид тварин, відсутній графічний інтерфейс, налаштування системи можливо виключно через програмний код.

Ще один представник застосунків для побудови симуляцій - Equilinox, реалізовано у форматі гри. Застосунок має різноманітні можливості впливу на систему, гарний візуальний стиль та зручне управління. Однак Equilinox є платним продуктом, має значні обмеження щодо вхідних даних, і сфокусований на процесі гри, а не аналізу отриманої інформації, тому орієнтований на іншу цільову аудиторію.

Отже, інтелектуальна система моделювання популяцій тварин та аналізу природного відбору, якій присвячена дана дипломна работа, доволі унікальна і поєднує у собі більшість переваг усіх наведених вище аналогів. Застосунки-аналоги мають набагато вужчий функціонал: зазвичай відсутня можливість аналізу даних, можливість налаштування системи, що значно звужує область застосування, або візуальна частина. Тобто наведених вище переваг достатньо, щоб на рівних конкурувати із вже існуючими продуктами.

1.3 Постановка задачі

1.3.1 **Призначення розробки**

Інтелектуальна система моделювання популяцій тварин та аналізу природного відбору призначена для спрощення роботи науковців у сферах популяційної біології та демекології.

1.3.2 **Цілі та задачі розробки**

Цілі створення системи моделювання популяцій та аналізу природного відбору:

- полегшення у роботі працівників у сфері популяційної біології;
- спрощення проведення популяційних досліджень;
- виявлення видів тварин, що мають ризик зникнути;
- полегшення прогнозування динаміки досліджуваного середовища;
- автоматизація проведення аналізу інформації.

Тобто основною ціллю створення додатку є спрощення контролю популяції тварин.

Для реалізації поставлених цілей система має вирішувати наступні задачі:

- створення та налаштування симуляції;
- надання можливості збереження та завантаження сесій користувача;
- проведення аналізу отриманих даних;
- відображення інформації про поточний стан видів.

Висновок до розділу

У ході написання розділу проаналізовано предметну область, виявлено вже існуючі аналоги, сформульовано постановку задачі та призначення системи.

В розділі описано предметне середовище, описана функціональна модель та визначені дійові особи.

Наведено діаграму варіантів використання, що показує функції, які може виконувати інтелектуальна система аналізу популяцій тварин та перелік

функціональних складових. Також наведена таблиця з функціональними вимогами щодо роботи застосунку за кожним варіантом використання.

В розділі описані наявні аналоги, проведено аналіз їх функціональних особливостей та недоліків. Проаналізовано переваги даного продукту над розробками конкурентів, обґрунтовано переваги даної системи над існуючими аналогами з технічної точки зору та з точки зору користувача.

Крім призначення системи, у розділі описані цілі розробки та наведені задачі для досягнення коректної роботи застосунку за вказаними у функціональній моделі вимогами.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

Усі вхідні дані надходять від користувача системи. На вхід дана система приймає статистичні дані щодо предмету дослідження. Користувач за допомогою графічного інтерфейсу налаштовує параметри системи відповідно даним для створення бажаної симуляції. Їх можна розділити на чотири умовні групи:

- дані про локацію (середовище);
- дані про рослинний світ;
- дані про види тварин;
- дані про початковий стан симуляції.

Дані про локацію відповідають за налаштування ландшафту, щільності рослинної фауни та кількість водних ресурсів.

Дані про рослинний світ використовуються для створення різновидів рослинної їжі та налаштування оновлення рослинних ресурсів.

Дані про види тварин відповідають за створення видів. Налаштування видів включає у себе задання користувачем назви виду, також системою при створенні кожному виду автоматично надається унікальний ID виду. Для функціонування візуального відображення реалізована можливість обрати 3d-модель для самців і самок виду. Раціон виду (травоїдний вид або хижий) задається за допомогою булевої змінної та доповнюється списком ID видів. Це дає змогу встановити інших тварин як їжу для виду хижаків (система харчових ланцюгів). Поведінка представників виду керується за допомогою таких даних як раціон; швидкість; стать; радіус реагування на середовище; відчуття голоду; відчуття посухи; потреба у їжі та воді; початок та кінець дітородного віку; час виношування нащадків; кількість нащадків; відношення самок до самців у виводку; вік смерті від старості; сила та максимальне значення відхилення від норми під час мутації, тощо.

Дані про початковий стан симуляції визначають початкову кількість особин кожної статі початкову кількість тварин певних видів. Перелік вхідних сигналів наведено у таблиці 2.1.

Таблиця 2.1 – Перелік реквізитів вхідних даних

Найменування	Кодове позначення	Тип, розрядність	Кодове позначення документів, що містять ці дані
Номер мапи ландшафту	mapID	int, 4 байт	MapUI.cs
Щільність перепон (камені, дерева, схили, тощо)	ObstacleFilling	float, 8 байт	MapUI.cs
Щільність рослинної фауни	TreeFilling	float, 8 байт	MapUI.cs
Назва виду рослини	speciesName	string	PlantUI.cs
Інтервал оновлення	spawnInterval	float, 8 байт	PlantUI.cs
Кількість особин при оновленні	spawnsPerInterval	float, 8 байт	PlantUI.cs
Назва виду тварин	speciesName	string	SpecieUI.cs
Швидкість особин виду	speed	float, 8 байт	SpecieUI.cs
Радіус реагування на об'єкти системи	senseRadius	float, 8 байт	SpecieUI.cs
Початок репродуктивного віку особин виду	startMatingAge	float, 8 байт	SpecieUI.cs
Кінець репродуктивного віку особин виду	endMatingAge	float, 8 байт	SpecieUI.cs

Продовження таблиці 2.1

Найменування	Кодове позначення	Тип, розрядність	Кодове позначення документів, що містять ці дані
Вік смерті від старості	deathAge	float, 8 байт	SpecieUI.cs
Потреба у їжі	hungerDecreasePerSecond	float, 8 байт	SpecieUI.cs
Потреба у воді	thirstDecreasePerSecond	float, 8 байт	SpecieUI.cs
Поріг відчуття посухи	thirstTrigger	float, 8 байт	SpecieUI.cs
Поріг відчуття голоду	hungerTrigger	float, 8 байт	SpecieUI.cs
Час вагітності	pregnantTime	float, 8 байт	SpecieUI.cs
Кількість нащадків	offspringsCount	int, 4 байт	SpecieUI.cs
Відсоток нащадків жіночої статі	femalePartInOfsprings	float, 8 байт	SpecieUI.cs
Відсоток нащадків чоловічої статі	offspringDevelopment	float, 8 байт	SpecieUI.cs
Шанс мутації генів	mutationChance	float, 8 байт	SpecieUI.cs
Сила мутації	mutationStrong	float, 8 байт	SpecieUI.cs
Максимальне відхилення при мутації	maxMutationStrong	float, 8 байт	SpecieUI.cs
Діапазон відхилення від середніх показників виду	Dispersion	float, 8 байт	PopulationSetting.cs

Кінець таблиці 2.1

Найменування	Кодове позначення	Тип, розрядність	Кодове позначення документів, що містять ці дані
Кількість особин виду чоловічої статі	MaleCount	int, 4 байт	PopulationSetting.cs
Кількість особин виду жіночої статі	FemaleCount	int, 4 байт	PopulationSetting.cs

2.2 Вихідні дані

Вихідними даними системи є симуляція, отримані з неї за допомогою графічного інтерфейсу користувача статистичні дані, та файл формату .txt з даними про початковий та кінцевий стани симуляції. Застосунок отримує нові дані за допомогою автоматизованої системи спостереження за симуляцією та за інтелектуальної системи аналізу даних;

Перелік вихідних сигналів наведено у таблиці 2.2.

Таблиця 2.2 – Перелік вихідних сигналів

Призначення	Найменування	Одиниці виміру	Діапазони зміни	Спосіб предст. інформації
Загальні дані про популяцію виду	Кількість особин виду	шт.	[0, max int]	Графічний інтерфейс користувача. Вікно аналізу. Вкладка “Загальні дані виду”.
	Динаміка розміру популяції	стан	Позитивна/ стабільна/ негативна	
	Середня тривалість життя	роки	[0, max float]	
	Біотичний потенціал виду	число	[0, max float]	

Продовження таблиці 2.2

Призначення	Найменування	Одиниці виміру	Діапазони зміни	Спосіб предст. інформації
Загальні дані про популяцію виду	Средне значення голоду	відсоток	[0, 100]	Графічний інтерфейс користувача. Вікно аналізу. Вкладка “Загальні дані виду”.
	Средне значення посухи	відсоток	[0, 100]	
Просторова структура популяції	Щільність популяції	особин на km²	[0, max float]	Графічний інтерфейс користувача. Вікно аналізу. Вкладка “Просторова структура”.
	Площа ареалу	km²	[0, max float]	
Статева структура популяції	Відсоток особин жіночої статі	відсоток	[0, 100]	Графічний інтерфейс користувача. Вікно аналізу. Вкладка “Статева структура”
	Відсоток особин чоловічої статі	відсоток	[0, 100]	
Вікова структура популяції	Середній вік	роки	[0, max float]	Графічний інтерфейс користувача. Вікно аналізу. Вкладка “Вікова структура”
	Молоді особини	відсоток	[0, 100]	
	Статевозрілі особини	відсоток	[0, 100]	
	Особини похилого віку	відсоток	[0, 100]	
	Вікова структура	тип	Регрессивний/ прогрессивний/ стаціонарний	

Кінець таблиці 2.2

Призначення	Найменування	Одиниці виміру	Діапазони зміни	Спосіб предст. інформації
Статистика смертей виду	Від посухи	відсоток	[0, 100]	Графічний інтерфейс користувача. Вікно аналізу. Вкладка “Статистика смертей виду”
	Від голоду	відсоток	[0, 100]	
	Від хижака	відсоток	[0, 100]	
	Від старості	відсоток	[0, 100]	
Середні показники представників виду	Перелік всіх характеристик	чисельні значення	[0, max float]	Графічний інтерфейс користувача. Вікно аналізу. Вкладка “Середні показники”
Генетична динаміка популяції	Перелік всіх характеристик	відсоток відхилення	[0, 100]	Графічний інтерфейс користувача. Вікно аналізу. Вкладка “Генетична динаміка популяції”
Пройшло часу від початку симуляції	День №	число	[0, max float]	Графічний інтерфейс користувача. Екран симуляції
Де max int ≈ 2 147 483 647; max float ≈ 3,4 x 10^{38}; max string ≈ 10^{9};				

Перелік вихідних документів наведено у таблиці 2.3.

Таблиця 2.3 – Перелік вихідних документів та їх реквізитів

Найменування	Кодове позначення	Перелік реквізитів	Перелік користувачів
Simulation.txt	savedData.cs	Усі дані з таблиць 2.1 та 2.2	Користувач системи

2.3 **Структура масивів інформації**

Після введення користувачем даних у застосунок та обробкою вводу системою UI, початкові дані зберігаються у оперативній пам’яті комп'ютера (RAM) та обробляються підсистемами застосунку. Для ефективної роботи з даними у підсистемах створюються колекції даних.

Колекція – це об'єкт у програмі, що містить у собі, таким чи іншим чином, набір значень одного або різних типів, та дозволяє звертатися до цих значень або записувати у себе нові. Призначення колекцій у даній системі - слугувати сховищем об'єктів і забезпечувати доступ до них. Також, більшість сучасних мов програмування забезпечують підтримку колекцій, мають спеціальний синтаксис і семантику стандартних операцій над ними.

Специфіка програми - велика кількість масивів даних одного типу, що потребують динамічних змін, визначає списки та асоціативні масиви (словники) найкращими різновидами колекцій для вирішення поставлених задач. Список - колекція з послідовним доступом та представляє собою зліченне число впорядкованих значень. Словник, навпроти, є неупорядкована колекція, що зберігає пари “ключ - значення” та надає доступ до елементів по ключу.

Події, що відбулися у симуляції, викликають функції підсистем, котрі у свою чергу заданим чином обробляють дані та надають інформацію про зміни користувачеві.

Схема концептуальна обробки даних зображена на рисунку 2.1 та наведена у графічному додатку.

Рисунок 2.1 – Схема концептуальна обробки даних

Висновок до розділу

В розділі детально описано вхідні та вихідні дані системи моделювання популяцій тварин. Наведено опис структур масивів інформації та обраних структур даних, побудована концептуальна схема обробки даних.

У вхідних даних описані джерела первісних даних, визначено яку саме інформацію та у якому вигляді потребує система. Наведена таблиці з переліком реквізитів вхідних даних, що містить у собі інформацію про найменування, кодове позначення, тип, розрядність вхідних даних та перелік кодових позначень документів, що містять ці дані.

У вихідних даних наведено перелік вихідних даних та вихідних документів. Також описано, що саме вони собою являють та наведені таблиці з переліком реквізитів вихідних даних, що містить у собі інформацію про призначення, найменування, одиниці виміру, діапазони зміни та спосіб представлення інформації.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Змістовна постановка задачі

Головна ціль розробки застосунку була визначена як спрощення контролю популяції тварин. Тобто змістовна постановка задачі розробки полягає у комплексному аналізі нових даних, отриманих з симуляції, не витрачаючи ресурси на дослідження реальних екосистем.

Аналіз має включати у себе визначення основних статистичних характеристик популяції видів тварин та виконання необхідних для аналізу розрахунків.

3.2 Математична постановка задачі

Задача полягає в розрахунку відсотків певних груп всередині популяції, в отриманні середніх показників особин видів та визначені таких характеристик популяцій тварин як динаміка розміру популяції, біотичний потенціал виду, щільність популяції, площа ареалу проживання, вікова структура та генетична динаміка популяції.

Вхідними даними аналітичної підсистеми є дані, отримані у результаті взаємодії об’єктів у середині симуляції, та початкові дані системи, введені користувачем системи. Їх детальний опис наведено у розділі 2 Інформаційного забезпечення.

3.3 Обґрунтування методу розв’язання

Більшість явищ у природному середовищі можно проаналізувати та виділити фактори впливу. Відкриті математично, закони розвитку, змін структур популяцій були підтверджені спостереженнями. У роботах [1] - [3] описані основні з них та наведені формули для обчислення показників популяції.

Для проведення аналізу, у даній роботі використовуються алгоритми побудови математичних моделей структур популяцій та формули популяційної біології.

3.4 Опис методів розв'язання

Розрахунок відсотка групи відносно популяції

Позначимо через x_{ij} відсоток представників групи *i* популяції *j* відносно розміру всієї популяції S_j , N_i - кількість особин групи *i.* Тоді:

$$x_{ij} = \frac{S_j}{N_i} \ , \tag{3.1}$$

Розрахунок середніх параметрів виду

Параметр виду (у даній системі) - характеристика окремої властивості особини виду. Наприклад, швидкість руху або вік смерті від старості.

Середне значення для кожного за показників виду буде обчислюватися за формулою:

$$x = \frac{1}{n}\sum_{i=1}^{n} x_i, \tag{3.2}$$

де *x* - середнє значення будь якої з характеристик,

n - кількість представників виду.

Визначення генетичної динаміки популяції

У біології, мутації — зміни генетичного матеріалу. У даній системі мутації реалізовані як зміна випадкового показника особини виду при народженні.

Характеристика генетичної динаміки популяції визначає міру розповсюдження мутації, тобто відхилення кожного з показників виду з початку створення симуляції. Відхилення показника визначається як різниця поточного середнього значення параметра виду та початкового значення того самого параметра виду.

Тобто для всіх параметрів виду генетична динаміка популяції буде обчислюватись як різниця векторів:

$$(x, y, z) = (x_1, y_1, z_1) - (x_2, y_2, z_2)\ , \qquad (3.3)$$

де x_1, y_1, z_1 - початкові значення параметрів виду,

x_2, y_2, z_2 - поточні середні значення параметрів виду,

вектор (x, y, z) - генетична динаміка популяції.

Визначення біотичного потенціалу виду

Приріст популяції виду завжди пропорційний кількості особин у ній, тобто якщо зростання популяції не обмежене зовнішними чинниками, популяція зростає прискорено (експоненціальний тип зростання популяції). У такому випадку динаміка зростання популяції визначається величиною біотичного потенціалу та описується диференціальним рівнянням А.Лотки:

$$\frac{dN}{dt} = rN\ , \qquad (3.4)$$

де N - чисельність популяції,

$\frac{dN}{dt}$ - зміна чисельності популяції (у даному випадку приріст),

r - репродуктивний (біотичний) потенціал.

Інший розвиток отримує ситуація в умовах обмежених харчових ресурсів. Лімітуючі фактори називають ємністю середовища. Ємність середовища визначається числом особин, потреби яких можуть бути задоволені ресурсами певного місцеперебування.

Смертність починає зростати, коли чисельність популяції досягає або тимчасово перевищує ємність екосистеми. Початковий експоненціальне зростання в початкових сприятливих умовах з часом поступово сповільнюється.

Біотичний або репродуктивний потенціал - це показник збільшення чисельності при відсутності лімітуючих факторів. Хоча в природних умовах біотичний потенціал ніколи не реалізується, його визначення необхідне при

розрахунку прогнозів зростання популяції або при розробці методів боротьби з шкідливими видами.

Біотичний потенціал розраховується за формулою:

$$r = b_t - d_t \,, \tag{3.5}$$

де b_t - число народжених,

d_t - число загиблих особин за один і той же період часу t,

t - максимальний час життя особини виду у даній системі.

Біотичний потенціал розраховується тільки для видів, для яких смертність від старості становить 100%.

Визначення динаміки розміру популяції

Для вимірювання чисельності популяції зазвичай використовують загальне число індивідів на території, або цензове число. Динаміка розміру популяції може бути трьох типів: позитивною, стабільною або негативною.

$$K = b_t - d_t \,, \tag{3.6}$$

де b_t - число народжених,

d_t - число загиблих особин за один і той же період часу t,

t - максимальний час життя особини виду у даній системі.

При $K > 0$ популяція зростає (позитивна динаміка), при $K < 0$ популяція вимирає (негативна динаміка), при $K = 0$ популяція стабільна.

Розрахунок площі ареалу проживання виду

Ареал — територія поширення виду. Розраховуэться за формулою:

$$S = (x_{max} - x_{min})(y_{max} - y_{min}), \quad (3.7)$$

де x_{max} - найбільший, x_{min} - найменший елемент у множині координат по *x* особин виду,

y_{max} - найбільший, y_{min} - найменший елемент у множині координат по *y* особин виду.

Визначення щільності популяції

Щільність популяції — це показник просторового розміщення співчленів популяції, що виражає число особин тварин у розрахунку на одиницю площі поверхні. Тобто:

$$D = \frac{S}{N}, \quad (3.8)$$

де S - площа ареалу проживання виду,

N - кількість особин виду,

D - щільність популяції.

Визначення типу вікової структури

Вікова структура популяції— розподіл особин популяції за віком.

$$K = N_{\text{м}} - N_{\text{с}}, \quad (3.9)$$

де $N_{\text{м}}$ - кількість молодих особин,

$N_{\text{с}}$ - кількість старих особин виду.

Особина вважається молодою до досягнення нею репродуктивного віку, та старою при його закінченні.

Тип вікової структури може бути регресивним ($K < 0$), прогресивним ($K > 0$) або стаціонарним ($K = 0$).

Висновок до розділу

В даному розділі були сформульовані змістовна та математична постановки задачі. Була вирішена задача аналізу отриманих у процесі роботи симуляції даних. Детально описані розрахунки та алгоритми, що використовує система. Також у розділі наведені основні терміни предметного середовища.

Можна зробити висновок, що розроблена система здатна до комплексного аналізу даних, та надає інформацію по усім найбільш значущим статистичним характеристикам популяцій тварин.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

Для розробки програмного продукту було обрано рушій гри Unity, оскільки це сучасний багатоплатформовий інструмент для створення тривимірних додатків. В Unity об'єднані різні програмні засоби, що використовуються при створенні програмного забезпечення - текстовий редактор, компілятор, відладчик і так далі. Рушій Unity дозвояє створівати додатки з гарфікою високої якості. При цьому, завдяки зручності використання, Unity робить створення застосунків та ігор максимально простим і комфортним, а мультиплатформеність рушія дозволяє охопити якомога більшу кількість платформ і операційних систем. Розроблені за допомогою засобів Unity застосунки працюють під усіма популярними операційними системами, такими як Windows, Linux, OS X, Apple iOS та Android.

Логіка у Unity пишеться за допомогою мови високого рівня C#. На сьогоднішній момент дана мова програмування одна з лідерів за швидкістю розвитку мов в IT-галузі. Мова C# поставляється з великою кількістю стандартних бібліотек, що значно прискорює розробку за рахунок використання готових рішень. Стандартні модулі надають засоби для роботи з різними типами файлів, системними викликами та мережевими з'єднаннями.

Разом з C# використовується об'єктно-орієнтований підхід до програмування. Це означає, що спочатку описуються абстрактні конструкції на основі предметної області, а потім реалізується взаємодія між ними. Даний підхід користується великою популярністю, тому що дозволяє працювати з підсистемами за принципом чорного ящика.

Також розробці використовувались такі патерни проектування:

Singleton — паттерн проектування, що дає гарантію існування лише одного екземпляру класу, та надає глобальний доступ до цього екземпляра.

Observer — паттерн проектування, що підтримує вільний зв'язок між об'єктами, які взаємодіють один з одним. При зміні стану одного з об'єктів всіх взаємодіючих з ним сповістять про цю подію.

Контроль версій виконувалася за допомгою системи контролю версій Git, що дозволяє зберігати стан та файли системи на кожній ітерації розробки. Git здобув популярність завдяки поєднаню таких переваг як легкість у використанні, швидкість та одночасно надання повного функціоналу сучасних систем контролю версій. Наприклад, підтримка нелінійної розробки та засоби для ефективної роботи з великими проектами.

4.2 Вимоги до технічного забезпечення

Вимоги до технічних засобів

Для коректної роботи розробленої системи моделювання популяцій тварин до складу технічних засобів повинні входити:

Комп’ютер з конфігурацією не гірше:

- Процессор: Dual Core від Intel або AMD 2.8 GHz;
- Об’єм оперативної пам’яті: 4 GB RAM;
- Відеокарта: Intel HD 3000;
- Storage: 450 MB available space.

Додатково на комп’ютер має бути встановлене програмне забезпечення:

- Операційна система Windows XP або новіша;
- DirectX: Version 9.0c.

Комп’ютерна периферія, до складу якої входить:

- Монітор;
- Клавіатура;
- Мишка.

4.2.1 Загальні вимоги

При роботі з системою, користувачі повинні дотримуватися правил експлуатації електронної обчислювальної техніки.

4.3 Архітектура програмного забезпечення

Був використаний такий архітектурний підхід – все застосування розділено на 5 окремих підсистем в залежності від їх змістовної функції.

Розглянемо кожну з них:

- **Підсистема інтерфейсу користувача (UI)** відповідає за взаємодію користувача з системою та відображення даних. Інтерфейс користувача реалізовано за допомогою інтрументів UnityEngine.UI;
- **Підсистема створення симуляції** відповідає за створення об’єктів на сцені Unity та їх налаштування. Також підсистема зв'язує об'єкти спеціального типу GameObject з 3D моделями, розробленими сценаріями на мові C# та компонентами рушія гри Unity;
- **Підсистема підтримки роботи симуляції** відповідає за регування системи на події всередині симуляції. Для відображення актуальної інформації на кожному кадрі, реалізована взаємодія даної підсистеми з підсистемою UI за допомгою паттерну Observer, що повідомляє підсистему інтерфейсу користувача про зміни у даних;
- **Підсистема аналізу даних** відповідає за обробку даних та статистичні розрахунки, що детально описані у розділі 3 Математичного забеспечення та за завантаження аналізу на локальний компьютер;
- **Підсистема збереження сесій користувача** відповідає за реалізацію функцій збереження та завантаження симуляцій. Тобто збір усіх даних симуляції, конвертацію цих даних для подальшого збереження та зберігання конвертованих даних.

Кожна з функцій підсистем застосунку реалізована у вигляді окремого класу, які підключаються як компоненти до об'єктів Unity. З цього витікає

значна перевагою системи - її модульність. Це означає, що всі шари застосування взаємодіють між собою через інтерфейси, не знаючи конкретних реалізацій один одного, що дає змогу легко проводити заміну на інші реалізації або додавати до системи нові компоненти.

Специфіка розробки у Unity потребує реалізації більшості класів через наслідування від класу MonoBehaviour. MonoBehaviour - базовий клас, з якого походить кожен сценарій Unity. У роботі [5] наведена детальна інформація щодо переваг використання наслідування від MonoBehaviour. Наприклад, можливість дочірнім класам використовувати такі методі як Start() – викликається при створенні екземпляру класу, Update() - викликається у кожному кадрі, та інші.

4.3.1 Діаграма класів

У діаграмі класів розглянемо класи усіх підсистем, оскільки вони є основою логіки застосунку та найкраще демонструють роботу системи. На Діаграмі класів представлені класи, інтерфейси, а також їхні відносини.

Діаграма класів зображена на рисунку 4.1 та наведена у графічному додатку.

Рисунок 4.1 – Діаграма класів системи

Рисунок 4.1 – Діаграма класів системи

4.3.2 Діаграма послідовності

На діаграмі послідовності розглянемо процес створення та налаштування ландшафту симуляції на сервісному рівні через класи. Діаграма відображає взаємодії об'єктів впорядкованих за часом. Також діаграма відображає задіяні для створення мапи об'єкти та послідовність відправлених повідомлень.

Об’єкти діаграми послідовності створення та налаштування ландшафту симуляції описані у таблиці 4.1.

Таблиця 4.1 – Об’єкти діаграми послідовності

Клас	Відповідальність
MapUI	Клас, що відповідає за відображення інтерфейсу налаштування ландшафту та обробку введених даних користовача.
MonoBeheviour	Клас, що надає можливість дочірнім класам відключатись до об’єктів Unity як компоненти та використовувати вбудовані методи.
Scanner	Клас, відповідальний за сканування мапи за допомогою методу Raycast та отримання списку координат, придатних для розміщення інших об’єктів. Сортування точок на придатні та ні залежить від тегу поверхні, на якій вони знаходяться.
MapManager	Клас, відповідальний за створення префабу мапи на сцені у Unity і за розрахунок розміщення дерев, схилів, каменів та інших об'єктів ландшафту. Також цей клас формує NavMesh-компонент мапи у Unity, що необхідний для коректного руху тварин у симуляції.

Структурна схема діаграми послідовності зображена на рисунку 4.2 та наведена у графічному додатку.

Рисунок 4.2 – Структурна схема діаграми послідовності

4.3.3 Діаграма компонентів

Дана діаграма компонентів показує розбиття розробленої системи на структурні компоненти та зв'язки (залежності) між цими компонентами.

На діаграмі задіяні об’єкти, представлені у таблиці 4.2.

Таблиця 4.2 – Об’єкти діаграми компонентів

Компонент	Опис компонента
UI	Відображення інтерфейсу та дії з ним
Simulation	Підтримка симуляції
Spawner	Додавання нових об'єктів у симуляцію
AnimalProperties	Налаштування поведінки тварин
MapManager	Створення мапи
Analytics	Аналіз даних
DataManager	Керування збереженнями у симуляції

Діаграма компонентів зображена на рисунку 4.3. та наведена у графічному додатку.

Рисунок 4.3 – Діаграма компонентів

4.3.4 **Специфікація функцій**

Найважливіші функції класів програмного забезпечення наведені в таблиці 4.3.

Таблиця 4.3 – Функції класів програмного забезпечення

Назва	Специфікація
Awake()	Використовується для ініціалізації будь-яких змінних або ігрового стану перед початком гри. Викликається лише один раз протягом життя екземпляра сценарію.
StaticRandom()	Використовується для розрахунку випадковостей альтернативним способом.
GetIndex()	Метод що повертає індекс елемента у списку.

Продовження табиці 4.3

Назва	Специфікація
Start()	Викликається коли сценарій увімкнено безпосередньо перед тим, як будь-який із методів оновлення буде викликаний вперше. Як і функція Awake(), Start() викликається рівно один раз у житті сценарію.
Update()	Метод викликається у кожному кадрі, якщо клас наслідується від MonoBeheviour.
DoStatsDispertion()	Випадково змінює початкові характеристики тварин.
SpawnFirstAnimals()	Створює на мапі перших тварин.
SetText()	Змінює текст на кнопці.
OnTriggerEnter()	Метод, що відповідає за реагування тварин на об’єкти (у полі зору).
OnTriggerExit()	Метод, що відповідає за реагування тварин на об’єкти (за межами поля зору).
Sense()	Надає “органи почуття” тварині.
CreateMap()	Створює локацію за заданими параметрами.
CreatePrefab()	Створює префаб, тобто оболонку для 3D-моделі.
SpawnPlant()	Розміщує рослини на локації на випадкових позиціях.
Exploring()	Задає поведінку тварини (подорож без цілі).
CheckIsHungry()	Перевірка чи голодна тварина.

Продовження табиці 4.3

Назва	Специфікація
CheckIsThirsty()	Перевірка чи відчуває спрагу тварина.
MoveToTarget()	Метод, що змушує об’єкт рухатися до заданої цілі.
RefreshStates()	Оновлення значень усіх параметрів.
SetPrefabs()	Задає 3D-моделі об’єкту.
GetCopy()	Створює копію об’єкту.
DoAfterCreation()	Виконує набір методів після створення обїєкту.
CheckIsReadyToMate()	Перевірка віку тварини.
GetGroundPositions()	Отримати позиції землі на поверхні мапи.
GetWaterPositions()	Отримати позиції води на поверхні мапи.
Scan()	Відсканувати локацію, щоб отримати списки позицій.
SetMap()	Задати мапу.
GetAllChilds()	Отримати список дочірніх елементів об’єкта.
GetPersent()	Отримати 3D-модель об’єкту.
GetRandomItem()	Повертає випадковий елемент списку.
Mating()	Створює нащадків за параметрами батьків.
LoadData()	Завантажує симуляцію.
SaveData()	Зберігає симуляцію.

Кінець таблиці 4.3

Назва	Специфікація
SaveAnalysis()	Зберігає аналіз симуляції.
GetDistanceTo()	Повертає значення відстані до об’єкту.

Висновок до розділу

У розділі детально описані засоби розробки використані у процесі створення застосування, обґрунтовано вибір цих засобів та наведені основні переваги кожного з них. Було наведено детальний опис усіх техічних рішень, які використовувались при розробці. Також у розділі описано, як саме функціонують всі його компоненти та які підходи необхідно використовувати при роботі з ними.

Описані вимоги до технічного забезпечення, які потрібні для використання та коректної роботи комплексу задач. Вимоги до технічного забезпечення включають перелік необхідних для запуску застосунку технічних вимог та загальні вимоги до експлуатації. Якщо середовище не відповідає вимогам, продукт не запуститься.

У розділі детально описана архітектура системи, її рівні та наведені деталі реалізації кожного з рівнів. Також описані переваги застосування, здобуті завдяки архітектурним рішенням.

У розділі наведено схеми опису архітектури системи та її компонентів: діаграму класів, діаграму послідовності, діаграму компонентів, таблиці з детальним описом об’єктів діаграм та наведено специфікацію функцій.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

Для початку роботи з програмою необхідно завантажити її на комп’ютер та запустити від імені адміністратору.

Після запуску застосунку буде відображена головна сторінка програми, на якій міститься інформація про розроблену систему та кнопки «Завантажити», «Створити» та «Х». Вигляд головного меню показано на рисунку 5.1.

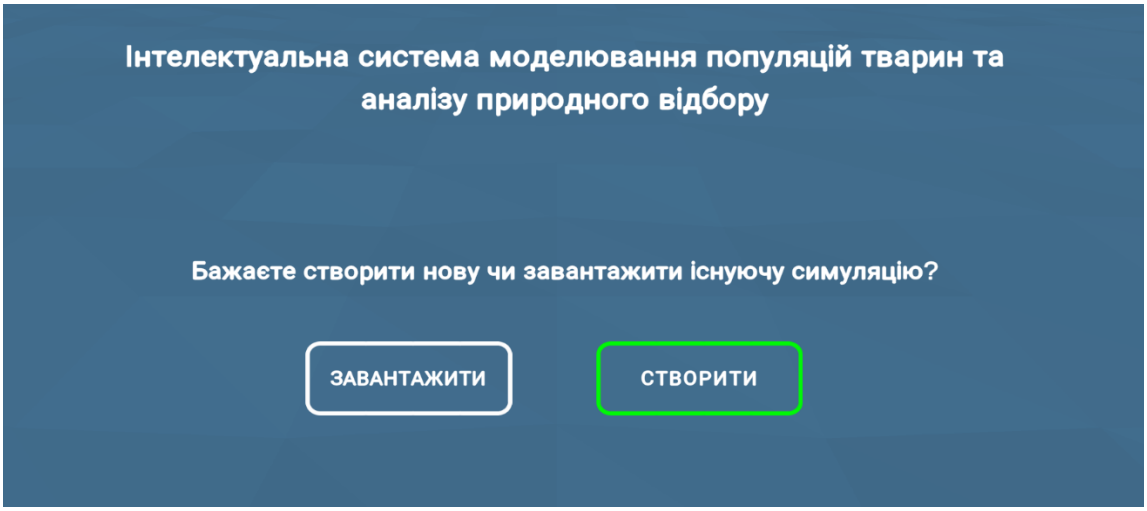

Рисунок 5.1 – Головне меню

Натискання кнопки «Завантажити» відображує екран списку збережених сессій, де користувач має змогу повернутися до головного меню, запустити або видалити зі списку обрану симуляцію. Вигляд вікна завантаження симуляції показано на рисунку 5.2.

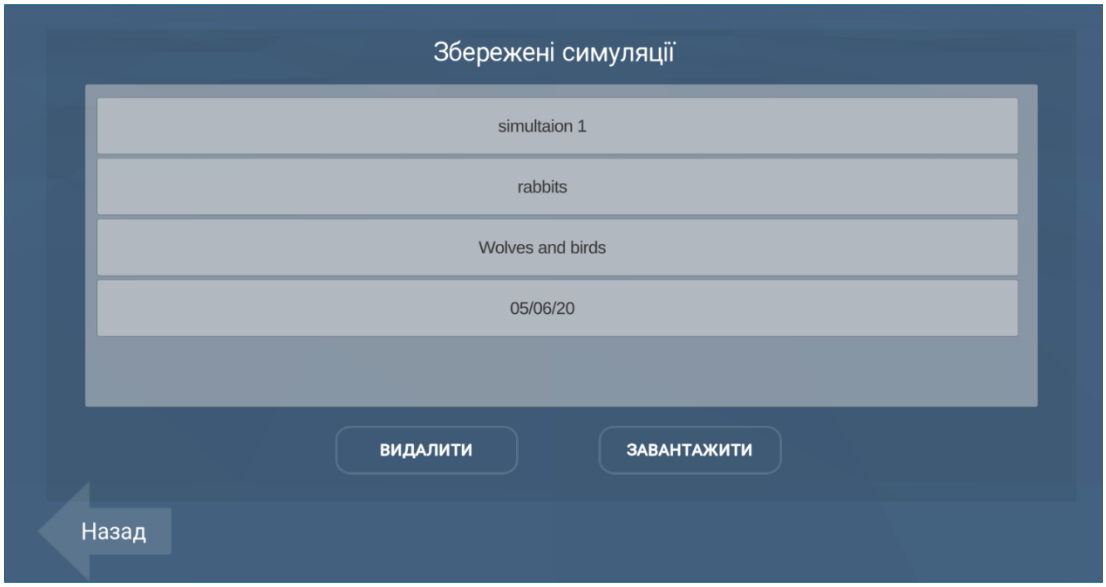

Рисунок 5.2 – Вікно завантаження симуляції

У випадку натискання кнопки «Створити» відображує екран налаштування ландшафту. Стрілки переключать варіанти водних мап, зміна інших параметрів (перепони, дерева, схили) виконується за допомогою повзунків. Кнопка «Далі» завершує редагування цього етапу.

Вигляд екрану налаштування ландшафту показано на рисунку 5.3.

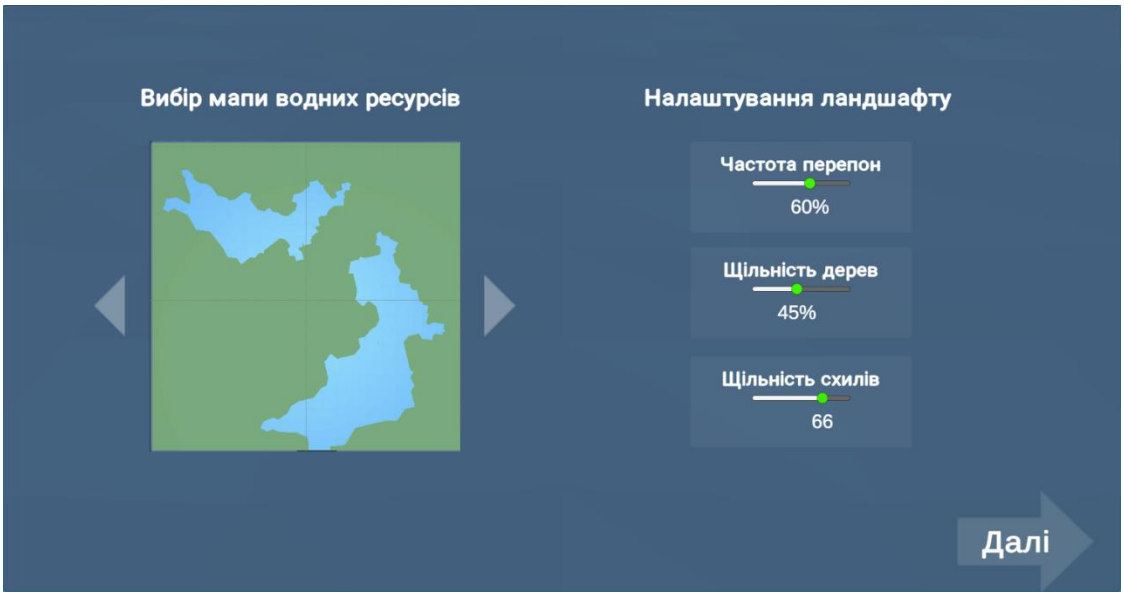

Рисунок 5.3 – Екран налаштування ландшафту

Далі користувач потрапляє на екран налаштування рослинної фауни.

Для створення виду необхідно натиснути кнопку «Додати» та ввести назву виду. Для видалення виду зі списку необхідно обрати вид у вікні «Види рослин» та натиснути кнопку «Видалити».

Вигляд екрану налаштування налаштування рослинної фауни показано на рисунку 5.4.

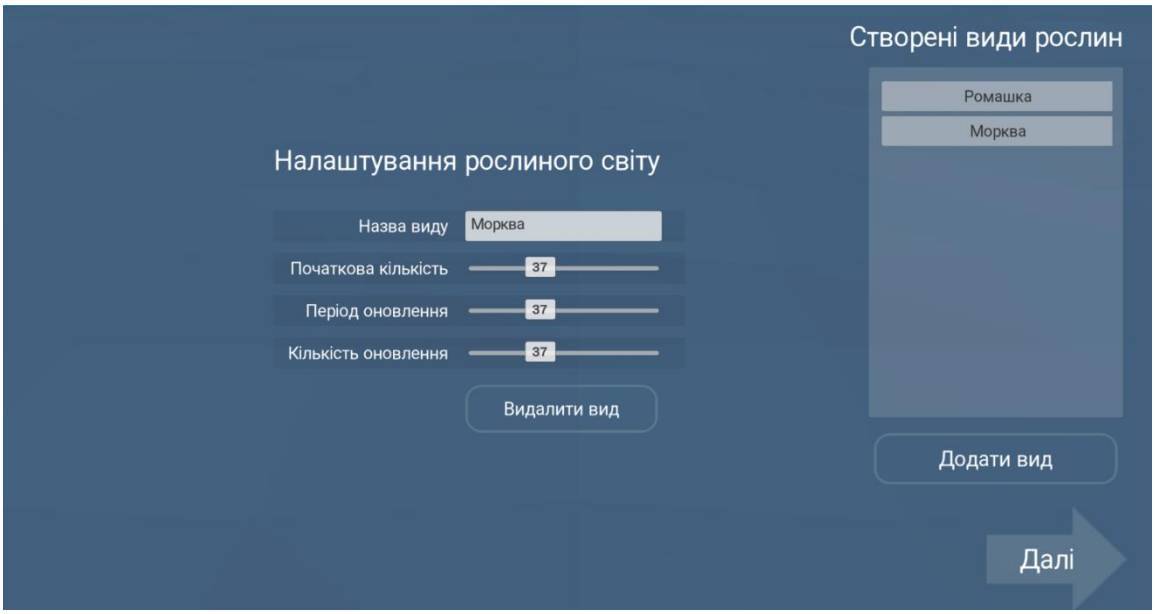

Рисунок 5.4 – Екран рослинної фауни

Далі користувач потрапляє на екран створення видів тварин. Для сворення нового виду необхідно спочатку натиснути кнопку «Додати вид» та ввести ім’я виду у текстове поле. Вигляд екрану створення видів тварин показано на рисунку 5.5.

Рисунок 5.5 – Екран створення видів тварин

Після створення видів на екрані користувача відобразиться екран налаштування початкових параметрів симуляції. Вигляд екрану початкових параметрів симуляції показано на рисунку 5.6.

Рисунок 5.6 – Екран початкових параметрів симуляції

Далі користувач потрапляє на екран спостереження за симуляцією. Керування камерою виконується за допомогою стрілок клавіатури. Кнопки

интерфейсу на цьому єкрані дозволять керувати часом, переглядати аналіз або повертатися до головного меню застосунку. Вигляд екрану спостереження за симуляцією показано на рисунку 5.7.

Рисунок 5.7 – Екран спостереження за симуляцією

Натискання на кнопку «зберегти» відкриває вікно збереження симуляції. Вигляд вікна збереження симуляції показано на рисунку 5.8.

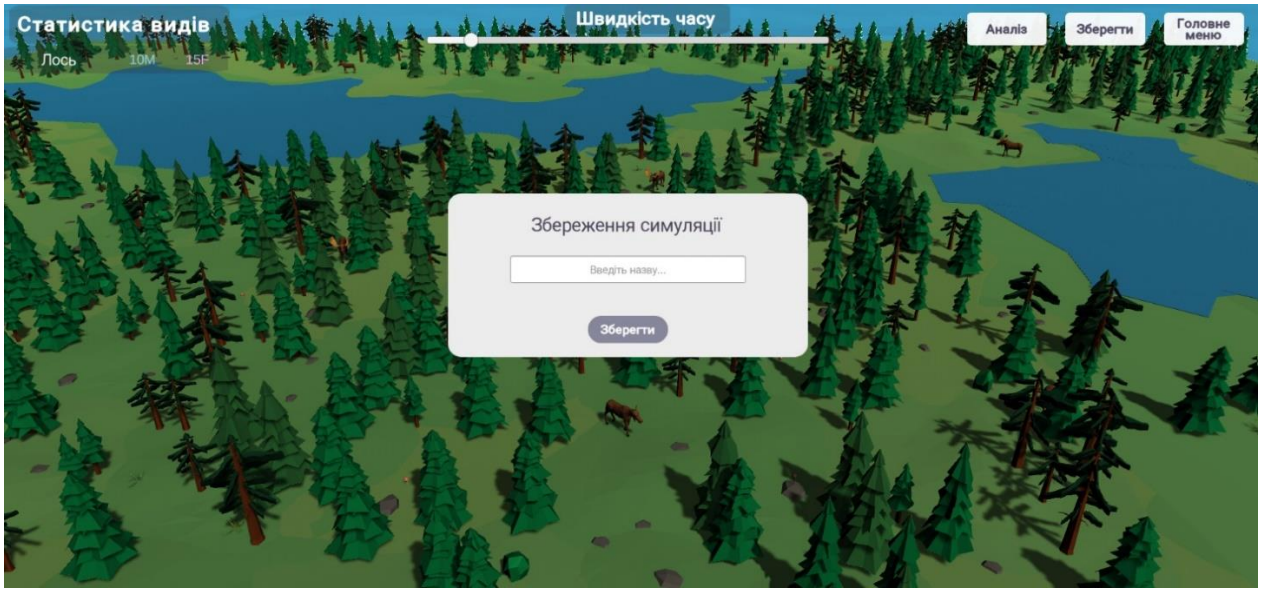

Рисунок 5.8 – Вікно збереження симуляції

Вікно аналізу надає можливість переглядати інформацію щодо кожного виду у списку. Кнопка «Завантажити» виконує функцію збереження файлу Simulation.txt на локальный компьютер. Вигляд вікна перегляду аналізу показано на рисунку 5.9.

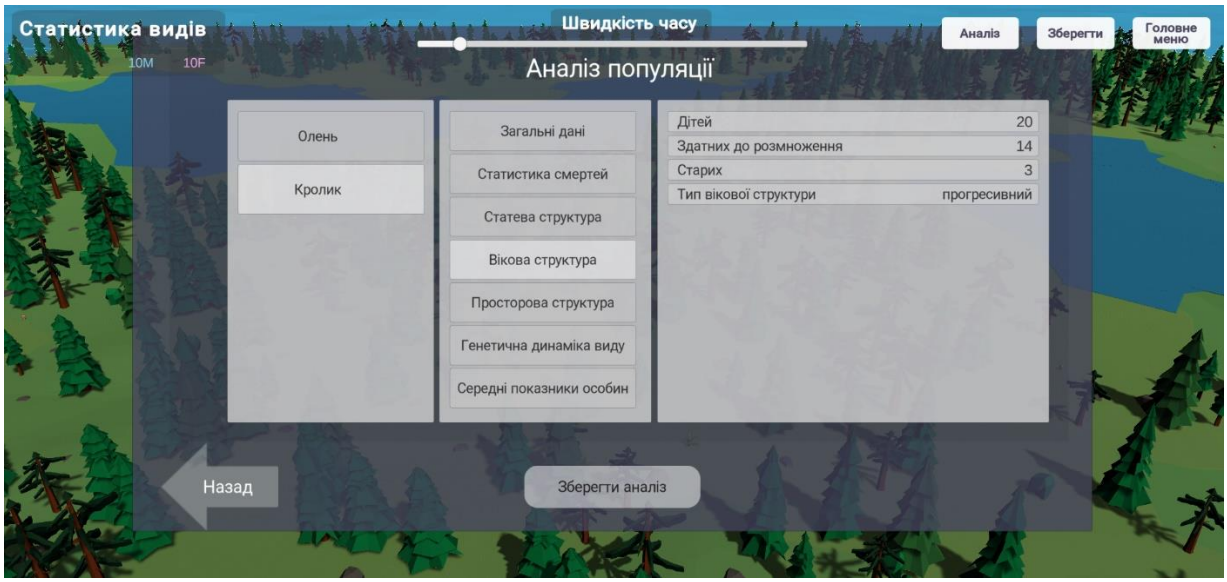

Рисунок 5.9 – Вікно перегляду аналізу

5.2 **Випробування програмного продукту**

В цьому підрозділі наведено опис проведених тестів і порядок їх виконання. Тестування проводилось для перевірки відповідності розробленого програмного забезпечення функціональним вимогам, представленим у технічному завданні на створення інтелектуальної системи моделювання популяцій тварин та аналізу природного відбору.

5.2.1 **Мета випробувань**

Метою випробувань являється перевірка відповідності функцій інтелектуальної системи моделювання популяцій тварин вимогам технічного завдання.

5.2.2 **Загальні положення**

Випробування проводяться на основі наступних документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3 Результати випробувань

В процесі випробувань програмного продукту була протестована вся функціональність розробленого застосунку.

У таблиці 5.1 наведений перелік випробувань основних функціональних можливостей та результати випробувань.

Таблиця 5.1 – Проведення випробувань

№ випроб.	Ситуація (умова)	Ефект (дія)	Ефект фактичний
1	Користувач вперше запускає програму	Користувач бачить екран, на якому відображене головне меню застосунку.	Співпадає з очікуваним
2	Користувач намагається створити нову симуляцію	Користувач натискає кнопку «Створити симуляцію» та потрапляє на екран налаштування ландшафту.	Співпадає з очікуваним
3	Користувач намагається завантажити симуляцію	Користувач натискає кнопку «Завантажити симуляцію» та бачить вікно завантаження симуляції. Обирає зі списку симуляцію та настискає «Завантажити». Після цього потрапляє на екран перегляду бажаної симуляції.	Співпадає з очікуваним

Продовження таблиці 5.1

№ випроб.	Ситуація (умова)	Ефект (дія)	Ефект фактичний
4	Користувач намагається видалити симуляцію	Користувач вибирає зі списку симуляцію та натискає кнопку «Видалити». Обрана симуляція видаляється зі списку.	Співпадає з очікуваним
5	Користувач налаштовує ландшафт	Користувач змінює параметри ландшафту за допомогою елементів графічного інтерфейсу.	Співпадає з очікуваним
6	Користувач налаштовує рослину фауну	Користувач налаштовує параметри рослинної фауни за допомогою елементів графічного інтерфейсу.	Співпадає з очікуваним
7	Користувач створює види	Користувач створює нові види тварин, ввівши назву виду та налаштувавши параметри видую.	Співпадає з очікуваним
8	Користувач налаштовує початкові параметри	Користувач змінює початкові параметри симуляції за допомогою елементів графічного інтерфейсу.	Співпадає з очікуваним
9	Користувач намагається керувати камерою у симуляції	Користувач вільно керує камерою за допомогою стрілок клавіатури.	Співпадає з очікуваним

Кінець таблиці 5.1

№ випроб.	Ситуація (умова)	Ефект (дія)	Ефект фактичний
10	Користувач намагається керувати часом у симуляції	Користувач зменшує значення повзунка керування часом, час у симуляції сповільнюється. При збільшенні значення - прискорюється.	Співпадає з очікуваним
11	Користувач зберігає симуляцію	При натисканні кнопки збереження симуляції відкривається вікно з полем вводу назви та кнопкою «Зберегти»	Співпадає з очікуваним
12	Користувач переглядає аналіз виду	Користувач переглядає аналіз популяції обраного зі списку виду за бажаним критерієм. Інформація відображається коректно.	Співпадає з очікуваним
13	Користувач намагається зберегти аналіз у файл	При натисканні кнопки збереження аналізу у папці з програмою створюється файл Simulation.txt	Співпадає з очікуваним

Висновок до розділу

У розділі наведено керівництво користувача, яке включає послідовний опис дій які можуть бути виконані користувачем у застосунку, а саме: перегляд списку збережених симуляцій, завантаження симуляції, створення нової симуляції, налаштування ландшафту, рослинної фауни та початкових умов нової симуляції, створення нових видів тварин, збереження існуючої та

видалення збереженої симуляції, перегляд статистичної інформації видів, керування часом та камерою у симуляції та збереження отриманого аналізу у файл.

Розділ містить у собі зображення екранних форм, опис роботи системи на цих формах для коректної роботи системи.

Також у розділі представлені сценарії проведених тестувань щодо відповідності функціональним вимогам. Застосунок успішно пройшов усі випробування, результати тестів співпали з функціональними вимогами до системи.

ЗАГАЛЬНІ ВИСНОВКИ

Під час виконання дипломного проекту було досліджено предметну область, визначено вимоги до застосування та доведена актуальність розробки. Визначені користувачі системи та варіанти використання застосунку. Виявлено необхідні вхідні дані та їх джерела. Наведено структуру та спосіб відображення вихідних даних. За допомогою діаграм діяльності та варіантів використання описано та зображено реалізовані у застосунку функції та процеси.

В процесі аналізу предметної області була сформульована задача розробки застосунку. Визначені основні цілі розробки інтелектуальної системи моделювання та аналізу популяцій тварин. Також визначено задачі, що повинні бути реалізовані у застосунку, та функціональні вимоги до системи.

Для розробки програмного забезпечення були використані такі засоби: рушій гри Unity, мова програмування високого рівня C#, паттерни проектування та система контролю версій Git. Наведено детальний опис засобів розробки та перелік основних переваг обраних засобів.

У дипломному проекті була розроблена архітектура програмного продукту, описані переваги застосування, здобуті завдяки архітектурним рішенням.

Також була розроблена модель обробки даних, що дозволяє надавати доступ до даних комплексу задач.

Наведена інструкція користувача по експлуатації системи моделювання популяцій тварин та аналізу природного відбору. Описані кроки, необхідні для виконання всіх функцій застосунку, наведені зображення екранних форм. Описані випробування програмного продукту на відповідність усім функціональним вимогам, які система успішно пройшла.

ПЕРЕЛІК ПОСИЛАНЬ

1. Сугаков В. Й. Основи синергетики / Володимир Йосипович Сугаков. – Київ: Обереги, 2001. – 287 с.
2. Корсак О. В. Основи сучасної екології : Навч. посіб. / О. В. Корсак, К. В. Плахотнік. – К: МАУП, 2020. – 340 с. – (4-те вид., перераб. і доп.).
3. Alan H. C. Population Biology: Concepts and Models / Hastings C. Alan. – New York: Springer, 1997. – 116 с.
4. Лановенко О. Г. Словник – довідник з екології : навч.-метод. посіб. / О. Г. Лановенко, О. О. Остапішина. – Херсон: ПП Вишемирський В.С., 2013. – 226 с.
5. Unity User Manual [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://docs.unity3d.com/.

ДОДАТОК А

Тексти програмного коду

Інтелектуальна система моделювання популяцій тварин та аналізу природного відбору

(Найменування програми (документа))

DVD-R

(Вид носія даних)

65 арк, 31 Mb

(Обсяг програми (документа) , арк.,) Mb)

Київ – 2020 року

DataManager.cs

```
using System.Collections;
using System.Collections.Generic;
using System.IO;
using NaturalSelectionTool;
using NaughtyAttributes;
using UnityEngine;


namespace NaturalSelection
{
   public class DataManager : MonoBehaviour
   {
      [SerializeField] FirstCreaturesManager creaturesManager;

      [Button]
      void SaveData()
      {
         var savedDatas = new
List<AnimalData>(creaturesManager.animals.Count);
         creaturesManager.animals.ForEach(x => savedDatas.Add(x.GetData()));

         var data = JsonHelper.ToJson<AnimalData>(savedDatas.ToArray(), true);
         File.WriteAllText("data.json", data);
      }

      [Button]
      void LoadData()
      {
         var data = new
List<AnimalData>(JsonHelper.FromJson<AnimalData>(File.ReadAllText("data.js
on")));
         creaturesManager.CreateAnimals(data);
      }
   }
}
```

CameraControler.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Cinemachine;

namespace NaturalSelection
{
   public class CameraController : MonoBehaviour
   {
      [SerializeField] Transform cameraLookAt;
      [SerializeField] float scrollSpeed = 15f;
      [SerializeField] float zoomSpeed = 9f;
      [SerializeField] CinemachineVirtualCamera runtimeCamera;
      CinemachineTransposer transposer;

      void Start()
      {
         transposer =
runtimeCamera.GetCinemachineComponent<CinemachineTransposer>();
      }

      void Update()
      {
         var scrollMod = Time.deltaTime * scrollSpeed;
         var zoomMod = Time.deltaTime * zoomSpeed;

         if (Input.GetKey(KeyCode.UpArrow))
         {
            if (Input.GetKey(KeyCode.Space))
            {
               var nextOffset = transposer.m_FollowOffset;
               nextOffset.y -= zoomMod;
               nextOffset.z += zoomMod;

               transposer.m_FollowOffset = nextOffset;
```

```
            }
            else
            {
                cameraLookAt.position += cameraLookAt.forward * scrollMod;
            }
        }

        if (Input.GetKey(KeyCode.DownArrow))
        {
            if (Input.GetKey(KeyCode.Space))
            {
                var nextOffset = transposer.m_FollowOffset;
                nextOffset.y += zoomMod;
                nextOffset.z -= zoomMod;

                transposer.m_FollowOffset = nextOffset;
            }
            else
            {
                cameraLookAt.position += -cameraLookAt.forward * scrollMod;
            }
        }

        if (Input.GetKey(KeyCode.LeftArrow))
        {
            cameraLookAt.position += -cameraLookAt.right * scrollMod;
        }

        if (Input.GetKey(KeyCode.RightArrow))
        {
            cameraLookAt.position += cameraLookAt.right * scrollMod;
        }
    }

}
}
```

FirstCreatureManager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using NaturalSelectionTool;
using System.IO;
using NaturalSelection.UI;
using System;
        namespace NaturalSelection
        {
            [Serializable]
            public class AnimalsList : List<AnimalProperties>
            {
            }

            [Serializable]
            public class FirstCreaturesManager : MonoBehaviour
            {
                [SerializeField] Scanner scanner;
                AnimalProperties defaultAnimalPrefab; //будет меняться
                [SerializeField] int femaleAnimalsCount; //будет меняться
                [SerializeField] int maleAnimalsCount; //будет меняться
                [SerializeField] float startStatsDispersion; //будет меняться
                [SerializeField] GameObject maleAnimalPrefab; //temp
                [SerializeField] GameObject femaleAnimalPrefab; //temp

                List<Vector3> spawnPositions;
                [SerializeField] public List<AnimalProperties> animals = new
List<AnimalProperties>();
                Spawner spawner;

                public AnimalsList Animals => animals;

                public void CreateAnimals(List<AnimalData> animalDatas)
                {
                    animalDatas.ForEach(x =>
                    {
```

```
                var animal = CreateEmptyAnimal();
                animal.propertyList = x.PropertyList;
                animal.SetPrefabs(maleAnimalPrefab, femaleAnimalPrefab);
                animals.Add(spawner.SpawnAnimal(animal, x.Position));
            });
        }

        void Start()
        {
            spawner = gameObject.AddComponent<Spawner>();
            scanner.Scanned += () => { spawnPositions = scanner.GetGroundPositions(); };

            var populationUI = UIManager.Get<SpeciePopulationUI>();

            populationUI.ContinuePressed += () =>
            {
                StartCoroutine(Wait());
                IEnumerator Wait()
                {
                    yield return new WaitUntil(() => spawnPositions != null && spawnPositions.Count > 0);

                    for (int i = 0; i < populationUI.SpecieButtons.Count; i++)
                    {
                        var popSettings = populationUI.PopulationSettings[i];
                        var inputPropList = new AnimalPropertyList(populationUI.SpecieButtons[i].PropertyList);
                        var max = (popSettings.MaleCount + popSettings.FemaleCount);
                        var specieAnimals = new List<AnimalProperties>(max);

                        for (int j = 0; j < max; j++)
                        {
                            var spawnPos = spawnPositions.GetRandomItem();
                            var newAnimal = CreateEmptyAnimal();

                            newAnimal.propertyList = inputPropList;
```

```
                        newAnimal.propertyList.isFemale = j < popSettings.MaleCount;
                        newAnimal.SetPrefabs(maleAnimalPrefab, femaleAnimalPrefab);

                        animals.Add(spawner.SpawnAnimal(newAnimal, spawnPos));
                        specieAnimals.Add(animals[animals.Count - 1]);
                    }

                    specieAnimals.ForEach(x => DoStatsDispersion(x, startStatsDispersion));
                }
            };
        };
    }

    AnimalProperties CreateEmptyAnimal() => new GameObject().AddComponent<AnimalProperties>();
    }
}
```

MapManager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using NaturalSelectionTool;
using NaturalSelection.UI;

namespace NaturalSelection
{

    public class MapManager : MonoBehaviour
    {
```

```
    GameObject map;
    [SerializeField] Scanner scanner;
    [SerializeField] Scanner hillsScanner;
    [SerializeField] float hillsFilling;
    [SerializeField] GameObject mapPrefab;

    [SerializeField] List<GameObject> treePrefabs = new
List<GameObject>();
    [SerializeField] List<GameObject> obstaclePrefabs = new
List<GameObject>();
    [SerializeField] List<GameObject> hillPrefabs = new
List<GameObject>();
    List<GameObject> entities = new List<GameObject>();
    List<Vector3> spawnPositions = new List<Vector3>();

    public GameObject Map => map;

    public Scanner Scanner => scanner;

    void Start()
    {
       UIManager.Get<SpeciePopulationUI>().ContinuePressed += () =>
       {
          CreateMap();
       };
    }

     void CreateMap()
    {
       if (map != null)
       {
          if (Application.isEditor)
             DestroyImmediate(map);
          else
             Destroy(map);

          entities.Clear();
       }
```

```
            map = Instantiate(mapPrefab);

            var treePercent = UIManager.Get<MapUI>().TreeFilling;
            var obstaclePercent = UIManager.Get<MapUI>().ObstacleFilling;
            var treeProbs = new List<float>() { treePercent, 100f - treePercent };
            var obstProbs = new List<float>() { obstaclePercent, 100f -
obstaclePercent };
            var hillProbs = new List<float>() { hillsFilling, 100f - hillsFilling };

            var hillsSpawnPoaitions =
hillsScanner.SetMap(map.transform.GetChild(0)).Scan().GetGroundPositions();

            hillsSpawnPoaitions.ForEach(x =>
            {
               if (hillProbs.RollDice() == 0)
                  CreatePrefab(hillPrefabs.GetRandomItem(), x, 0.1f);
            });

            Scanner.SetMap(map.transform.GetChild(0)).Scan();

            spawnPositions = Scanner.GetGroundPositions();
            spawnPositions.ForEach(x =>
            {
                if (treeProbs.RollDice() == 0)
                  CreatePrefab(treePrefabs.GetRandomItem(), x, 1.3f);

               else if (obstProbs.RollDice() == 0)
                  CreatePrefab(obstaclePrefabs.GetRandomItem(), x, 2.5f);
            });
         }

         void CreatePrefab(GameObject prefab, Vector3 spawnPos, float
posOffsetMult)
         {
            var randomOffset = UnityEngine.Random.insideUnitSphere *
posOffsetMult;
            randomOffset.y = 0;
```

```
                var randomRot = StaticRandom.Instance.Next(0, 150);
                var rotation = Quaternion.Euler((prefab.transform.localRotation.eulerAngles.x), randomRot, (prefab.transform.localRotation.eulerAngles.z));
                var position = spawnPos + randomOffset + prefab.transform.localPosition;

                var entity = Instantiate(prefab, position, rotation);
                entities.Add(entity);
                entity.transform.SetParent(map.transform);
            }
        }
    }
```

PlantSpawner.cs

```
using System.Collections;
using System.Collections.Generic;
using NaturalSelection.UI;
using NaturalSelectionTool;
using UnityEngine;
using NaturalSelectionTool;

namespace NaturalSelection
{
    public class PlantSpawner : MonoBehaviour
    {
        [SerializeField] MapManager mapManager;
        [SerializeField] List<GameObject> plantFoodPrefabs = new List<GameObject>();
        [SerializeField] Scanner scanner;
        List<GameObject> plants = new List<GameObject>();
        Transform foodParent;

        void Start()
        {
            foodParent = new GameObject().transform;
```

```
            scanner.Scanned += () =>
            {
                UIManager.Get<PlantsUI>().SpecieButtons.ForEach(x =>
                {
                    for (int i = 0; i < x.PropertyList.initialCount; i++)
                    {
                        CreatePrefab(plantFoodPrefabs.GetRandomItem(),
scanner.GetGroundPositions().GetRandomItem(), 1.5f);
                    }

                    StartCoroutine(Wait());
                    IEnumerator Wait()
                    {
                        yield                                                    return
Get.DelayInSeconds(x.PropertyList.spawnInterval);

                        for (int i = 0; i < x.PropertyList.spawnsPerInterval; i++)
                        {
                            CreatePrefab(plantFoodPrefabs.GetRandomItem(),
scanner.GetGroundPositions().GetRandomItem(), 1.5f);
                        }

                        StartCoroutine(Wait());
                    }
                });
            };

            void CreatePrefab(GameObject prefab, Vector3 spawnPos, float
posOffsetMult)
            {
                var randomOffset = UnityEngine.Random.insideUnitSphere *
posOffsetMult;
                randomOffset.y = 0;

                var randomRot = StaticRandom.Instance.Next(0, 150);
```

```
                var rotation = Quaternion.Euler((prefab.transform.localRotation.eulerAngles.x), randomRot, (prefab.transform.localRotation.eulerAngles.z));
                var position = spawnPos + randomOffset + prefab.transform.localPosition;

                var entity = Instantiate(prefab, position, rotation);
                plants.Add(entity);
                entity.transform.SetParent(foodParent);
            }
        }
    }
}
```

ReferenceHolder.cs

```
using System.Collections;
using System.Collections.Generic;
using NaturalSelection;
using UnityEngine;

namespace NaturalSelection
{
    public class ReferenceHolder : Singleton<ReferenceHolder>
    {
        [SerializeField] Transform uiWindowsParent;

        public Transform UIWindowsParent => uiWindowsParent;
    }
}
```

TimeManager.cs

```
using System.Collections;
using System.Collections.Generic;
using NaturalSelection.UI;
```

```
using UnityEngine;

namespace NaturalSelection
{
    public class TimeManager : MonoBehaviour
    {

        void Start()
        {
            UIManager.Get<RuntimeUI>().TimeScaleChanged += (value) =>
            {
                Time.timeScale = value;
            };
        }
    }
}
```

UIManager.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using NaturalSelection.UI;
using NaturalSelectionTool;
using UnityEngine;

namespace NaturalSelection
{
    public static class UIManager
    {
        static Dictionary<Type, UIWindow> uiWindows = new
Dictionary<Type, UIWindow>();
        static Validator validator = new Validator(() => uiWindows.Count > 0);


[RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.AfterSceneLoad)]
        static void Init()
```

```
            {
                var uiParent = ReferenceHolder.Instance.UIWindowsParent;

                for (int i = 0; i < uiParent.transform.childCount; i++)
                {
                    var uiWindow = uiParent.transform.GetChild(i).GetComponent<UIWindow>();

                    if (uiWindow != null)
                        uiWindows.Add(uiWindow.GetType(), uiWindow);
                }

                foreach (var window in uiWindows)
                    window.Value.gameObject.SetActive(true);

                // Get<LoadingUI>().FadedIn += () => { GC.Collect(); };

                validator.Check();
            }

            public static T Get<T>() where T : UIWindow => uiWindows[typeof(T)] as T;

            public static void DoAfterValidate(Action action)
            {
                validator.DoAfterValidate(action);
            }
        }
    }
```

AnimalProperties.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
```

```
using NaturalSelectionTool;
using System;

namespace NaturalSelection
{
    [Serializable]
    public class AnimalProperties : MonoBehaviour
    {
        public List<int> spiciesIdToEat { get; set; }
        public NavMeshAgent navMeshAgent;
        public SphereCollider coll;

        GameObject maleAnimalPrefab; //temp
        GameObject femaleAnimalPrefab; //temp

        const float maxHunger = 100;
        const float maxThirst = 100;


        [SerializeField] public AnimalPropertyList propertyList;


        [SerializeField] bool isHungry = false;
        [SerializeField] bool isThirsty = false;
        [SerializeField] bool isReadyToMate = false;
        [SerializeField] bool isSet;
        [SerializeField] float currentAge = 0f;
        [SerializeField] float currentHunger = 100f;
        [SerializeField] float currentThirst = 100f;
        [SerializeField] string currentAction = "Exploring";

        //Нужно для спавнера?
        public void SetPrefabs(GameObject maleAnimalPrefab, GameObject
femaleAnimalPrefab)
        {
            this.maleAnimalPrefab = maleAnimalPrefab;
            this.femaleAnimalPrefab = femaleAnimalPrefab;
        }
```

```
    public AnimalData GetData()
    {
        return new AnimalData(propertyList, transform.position,
transform.rotation);
    }

    //Проверки состояний
    bool CheckIsHungry()
    {
        return (propertyList.hungerTrigger > currentHunger);
    }

    bool CheckIsThirsty()
    {
        return (propertyList.thirstTrigger > currentThirst);
    }

    bool CheckReadyToMate()
    {
        bool inOkAge = (currentAge > propertyList.startMatingAge) &&
(currentAge < propertyList.endMatingAge);
        return (inOkAge && !isHungry && !isThirsty);
    }

    void RefreshStates()
    {
        isReadyToMate = CheckReadyToMate();
        isThirsty = CheckIsThirsty();
        isHungry = CheckIsHungry();
    }

    void MoveToTarget(Vector3 target)
    {
        if (NavMesh.SamplePosition(target, out var hit, 10, 1))
            navMeshAgent.SetDestination(hit.position);
    }
```

```
        void Exploring()
        {
            if (navMeshAgent.remainingDistance < 2)
            {
                currentAction = "Exploring";
                var randomOffset = UnityEngine.Random.insideUnitSphere * 20 +
transform.position;
                randomOffset.y = 0;
                MoveToTarget(randomOffset);
            }
        }

        public GameObject GetPrefab()
        {
            return propertyList.isFemale ? femaleAnimalPrefab :
maleAnimalPrefab;
        }

        public void DoAfterCreation()
        {
            isSet = true;
            navMeshAgent = GetComponent<NavMeshAgent>();
            coll = GetComponent<SphereCollider>();
            var prefab = Instantiate(GetPrefab(), transform);
        }

        void Awake()
        {
            propertyList = new AnimalPropertyList();
        }

        void FixedUpdate()
        {
            if (!isSet)
                return;

            RefreshStates();
            Exploring();
```

```
        }
    }

}
```

Scanner.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace NaturalSelection
{
    public class Scanner : MonoBehaviour
    {
        public event Action Scanned;
        Transform map;
        [SerializeField] float border = 2.1f;
        [SerializeField] float scanFrequencyX = 0.015f;
        [SerializeField] float scanFrequencyY = 0.015f;
        List<Vector3> groundPositions = new List<Vector3>();
        List<Vector3> waterPositions = new List<Vector3>();


        public Scanner SetMap(Transform map)
        {
            this.map = map;
            return this;
        }

        void Awake()
        {
 groundPositions.Clear();
            waterPositions.Clear();
        }
```

```
public Scanner Scan()
{
   var x = map.transform.localScale.x / border;
   var mapForward = map.transform.forward;
   var mapRight = map.transform.right;

   var bound1 = (mapForward * x) + (mapRight * x);
   var bound2 = (mapForward * x) + (-mapRight * x);
   var bound3 = (-mapForward * x) + (-mapRight * x);
   var bound4 = (-mapForward * x) + (mapRight * x);

   var scanPos = bound1;
   var scanStartPos = bound1;
   var scanEndPos = bound2;
   var heightOffset = new Vector3(0, 5, 0);
   groundPositions.Clear();
   waterPositions.Clear();

   for (var t = 0f; t < 1; t += scanFrequencyX)
   {
      for (var t2 = 0f; t2 < 1; t2 += scanFrequencyY)
      {
         scanPos = Vector3.Lerp(scanStartPos, scanEndPos, t2);

         if (Physics.Raycast(scanPos, Vector3.down, out var hit, 50))
         {
            if (hit.transform.gameObject.CompareTag("Ground"))
            {
               groundPositions.Add(hit.point);
            }

            if (hit.transform.gameObject.CompareTag("Water"))
            {
               waterPositions.Add(hit.point);
            }
         }
      }
```

```
                scanStartPos = Vector3.Lerp(bound1, bound4, t) + heightOffset;
                scanEndPos = Vector3.Lerp(bound2, bound3, t) + heightOffset;
            }

            Scanned?.Invoke();
            return this;
        }

        public List<Vector3> GetGroundPositions()
        {
            if (groundPositions.Count == 0)
            {
                Scan();
            }

            return groundPositions;
        }

        public List<Vector3> GetWaterPositions()
        {
            if (waterPositions.Count == 0)
            {
                Scan();
            }

            return waterPositions;
        }
    }

}
```

Spawner.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using NaturalSelectionTool;
```

```
using UnityEngine.AI;

namespace NaturalSelection
{
    public class Spawner : MonoBehaviour
    {
        GameObject animalHolder;

        public AnimalProperties SpawnAnimal(AnimalProperties animal,
Vector3 pos)
        {
            if (animalHolder == null)
            {
                animalHolder = new GameObject();
                animalHolder.name = "Animal Holder";
            }

            var randomRot = StaticRandom.Instance.Next(0, 150);
            //          var          creature          =
Instantiate(animal.gameObject).GetComponent<AnimalProperties>();

            animal.transform.position = pos;
            animal.transform.rotation = Quaternion.Euler(0, randomRot, 0);
            animal.transform.SetParent(animalHolder.transform);

            animal.gameObject.AddComponent<NavMeshAgent>();
            animal.gameObject.AddComponent<SphereCollider>();

            animal.DoAfterCreation();

            return animal;
        }
    }
}
```

Sence.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
using NaturalSelectionTool;

namespace NaturalSelection
{
    public class Sense : MonoBehaviour
    {
        public List<GameObject> nearFood = new List<GameObject>();
        public List<GameObject> nearMates = new List<GameObject>();
        public List<GameObject> nearWater = new List<GameObject>();
        public List<GameObject> nearEnemies = new List<GameObject>();

        AnimalProperties animalProperties;
        float senseRadius;
        List<int> spiciesIdToEat;
        int speciesId;
        bool isFemale;

        public void Add(AnimalProperties animalProperties)
        {
            this.animalProperties = animalProperties;
            this.senseRadius = animalProperties.propertyList.senseRadius;
            spiciesIdToEat = animalProperties.spiciesIdToEat;
            speciesId = animalProperties.propertyList.speciesId;
            isFemale = animalProperties.propertyList.isFemale;
        }

        void OnTriggerEnter(Collider someColl)
        {
            var someObject = someColl.gameObject;
            var animalRefs = someObject.GetComponent<AnimalProperties>();

            if (animalRefs == null)
            {
                if (someObject.CompareTag("Water"))
```

```
                {
                    nearWater.Add(someObject);
                }
                else if (someObject.CompareTag("PlantFood") &&
spiciesIdToEat.Count == 0)
                {
                    nearFood.Add(someObject);
                }
            }
            else if (speciesId == animalRefs.propertyList.speciesId &&
isFemale != animalRefs.propertyList.isFemale)
            {
                nearMates.Add(someObject);
            }
            else if (spiciesIdToEat.Contains(animalRefs.propertyList.speciesId))
            {
                nearFood.Add(someObject);
            }
            else if (animalRefs.spiciesIdToEat.Contains(speciesId))
            {
                nearEnemies.Add(someObject);
            }
            Debug.Log("From OnTriggerEnter()");
        }

        void OnTriggerExit(Collider someColl)
        {
            var someObject = someColl.gameObject;
            var animalRefs = someObject.GetComponent<AnimalProperties>();
            if (animalRefs == null)
            {
                if (someObject.CompareTag("Water"))
                {
                    nearWater.Remove(someObject);
                }
                else if (someObject.CompareTag("PlantFood") &&
spiciesIdToEat.Count == 0)
                {
```

```
                nearFood.Remove(someObject);
            }
        }
        else if (speciesId == animalRefs.propertyList.speciesId)
        {
            nearMates.Remove(someObject);
        }
        else if (spiciesIdToEat.Contains(animalRefs.propertyList.speciesId))
        {
            nearFood.Remove(someObject);
        }
        else if (animalRefs.spiciesIdToEat.Contains(speciesId))
        {
            nearEnemies.Remove(someObject);
        }
    }

  }
}
```

AnimalPropertyList.cs

```
using System;
using UnityEngine;

namespace NaturalSelection
{
  [Serializable]
  public class AnimalPropertyList
  {
    [SerializeField] public int speciesId;
    [SerializeField] public string speciesName = "noname";
    [SerializeField] public bool isFemale;
    [SerializeField] public int speed;
    [SerializeField] public float senseRadius;
    [SerializeField] public float startMatingAge;
    [SerializeField] public float endMatingAge;
```

```
        [SerializeField] public float deathAge;
        [SerializeField] public float hungerDecreasePerSecond;
        [SerializeField] public float thirstDecreasePerSecond;
        [SerializeField] public float thirstTrigger;
        [SerializeField] public float hungerTrigger;
        [SerializeField] public float pregnantTime;
        [SerializeField] public int offspringsCount;
        [SerializeField] public float femalePartInOfsprings;
        [SerializeField] public float offspringDevelopment; //"Развитость"
потомства, какой процент от своих будущих статов они получают
        [SerializeField] public float mutationChance;
        [SerializeField] public float mutationStrong;
        [SerializeField] public float maxMutationStrong;

        public AnimalPropertyList()
        {

        }

        public AnimalPropertyList(AnimalPropertyList original)
        {
            speciesId = original.speciesId;
            speciesName = original.speciesName;
            isFemale = original.isFemale;
            speed = original.speed;
            senseRadius = original.senseRadius;
            startMatingAge = original.startMatingAge;
            endMatingAge = original.endMatingAge;
            deathAge = original.deathAge;
            hungerDecreasePerSecond = original.hungerDecreasePerSecond;
            thirstDecreasePerSecond = original.thirstDecreasePerSecond;
            thirstTrigger = original.thirstTrigger;
            hungerTrigger = original.hungerTrigger;
            pregnantTime = original.pregnantTime;
            offspringsCount = original.offspringsCount;
            femalePartInOfsprings = original.femalePartInOfsprings;
            offspringDevelopment = original.offspringDevelopment;
            mutationChance = original.mutationChance;
```

```
            mutationStrong = original.mutationStrong;
            maxMutationStrong = original.maxMutationStrong;
        }
    }
}
```

PlantPropertyList.cs

```
using System;
using UnityEngine;

namespace NaturalSelection
{
    [Serializable]
    public class PlantPropertyList
    {
        [SerializeField] public int speciesId;
        [SerializeField] public string speciesName = "noname plant";
        [SerializeField] public float spawnInterval;
        [SerializeField] public float spawnsPerInterval;
        [SerializeField] public float initialCount;

        public PlantPropertyList()
        {

        }

        public PlantPropertyList(PlantPropertyList original)
        {
            speciesId = original.speciesId;
            speciesName = original.speciesName;
            spawnInterval = original.spawnInterval;
            spawnsPerInterval = original.spawnsPerInterval;
            initialCount = original.initialCount;
        }
    }
}
```

SavedData.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace NaturalSelection
{
    [Serializable]
    public class AnimalData
    {
        [SerializeField] AnimalPropertyList propertyList;
        [SerializeField] Vector3 position;
        [SerializeField] Quaternion rotation;

        public AnimalData(AnimalPropertyList propertyList, Vector3 position,
Quaternion rotation)
        {
            this.propertyList = propertyList;
            this.position = position;
            this.rotation = rotation;
        }

        public Quaternion Rotation => rotation;
        public Vector3 Position => position;
        public AnimalPropertyList PropertyList => propertyList;
    }

    [Serializable]
    public class PlantData
    {
        [SerializeField] PlantPropertyList propertyList;
        [SerializeField] Vector3 position;
        [SerializeField] Quaternion rotation;
```

```
            public PlantData(PlantPropertyList propertyList, Vector3 position,
Quaternion rotation)
            {
               this.propertyList = propertyList;
               this.position = position;
               this.rotation = rotation;
            }

            public Quaternion Rotation  => rotation;
            public Vector3 Position  => position;
            public PlantPropertyList PropertyList  => propertyList;
         }
      }
```

MapUI.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System;
using TMPro;

namespace NaturalSelection.UI
{
   public class MapUI : UIWindow
   {
      public event Action ContinueButtonPressed;

      public float ObstacleFilling => obstalceFillingSlider.value;
      public float TreeFilling => treeFillingSlider.value;

      [SerializeField] Button continueButton;
      [SerializeField] Slider obstalceFillingSlider;
      [SerializeField] Slider treeFillingSlider;
      [SerializeField] TextMeshProUGUI obstacleFillingValue;
      [SerializeField] TextMeshProUGUI treeFillingValue;
```

```
        void Start()
        {
            obstacleFillingValue.text = $"{obstalceFillingSlider.value}%";
            treeFillingValue.text = $"{treeFillingSlider.value}%";

            obstalceFillingSlider.onValueChanged.AddListener((value) =>
            {
                obstacleFillingValue.text = $"{value}%";
            });

            treeFillingSlider.onValueChanged.AddListener((value) =>
            {
                treeFillingValue.text = $"{value}%";
            });

            continueButton.onClick.AddListener(() =>
            {
                ContinueButtonPressed?.Invoke();
                Hide();
            });

            UIManager.Get<PreviewUI>().ContinueButtonPressed += () =>
Show();

            Hide();
        }
    }
}
```

PlantSpecieButton.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using NaturalSelection;
using TMPro;
```

```
using UnityEngine;
using UnityEngine.UI;

namespace NaturalSelection
{
   public class PlantSpecieButton : MonoBehaviour
   {
      public event Action Pressed;
      public event Action<string> NameChanged;

      public string SpecieName => specieName.text;

      public PlantPropertyList PropertyList { get => propertyList; set => propertyList = value; }

      [SerializeField] Button button;
      [SerializeField] TextMeshProUGUI specieName;
      PlantPropertyList propertyList;


      public void SetProperies(PlantPropertyList properties)
      {
         propertyList = properties;
      }

      void Start()
      {
         button.onClick.AddListener(() => Pressed?.Invoke());
      }

      public void SetText(string text)
      {
         specieName.text = text;
         NameChanged?.Invoke(text);
      }
   }
}
```

PlantsUI.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using NaturalSelectionTool;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace NaturalSelection.UI
{
   public class PlantsUI : UIWindow
   {
      public event Action ContinueButtonPressed;
      public event Action AddPressed;
      public event Action<PlantSpecieButton> SpecieAdded;
      public event Action<int> RemovePressed;
      public event Action SelectedSpecieChanged;

      [SerializeField] Slider initialCount;
      [SerializeField] Slider spawnInterval;
      [SerializeField] Slider spawnPerInterval;
      [SerializeField] TMP_InputField specieNameInputField;
      [SerializeField] Button continueButton;
      [SerializeField] Button addButton;
      [SerializeField] Button removeButton;
      [SerializeField] Transform speciesParent;
      [SerializeField] GameObject plantSpecieButtonPrefab;

      List<PlantSpecieButton> specieButtons = new
List<PlantSpecieButton>();
      PlantSpecieButton selectedSpecie;

      public List<PlantSpecieButton> SpecieButtons => specieButtons;

      void Start()
      {
```

```
        continueButton.onClick.AddListener(() =>
        {
            SetPropertiesFormSliders(selectedSpecie.PropertyList);
            ContinueButtonPressed?.Invoke();
            Hide();
        });

        addButton.onClick.AddListener(() =>
        {
            AddSpecie();
        });

        removeButton.onClick.AddListener(() =>
        {
            RemovePressed?.Invoke(99999);
        });

        specieNameInputField.onValueChanged.AddListener((value) =>
        {
            selectedSpecie.SetText(value);
        });

        UIManager.Get<MapUI>().ContinueButtonPressed += () =>
{ Show(); };

        AddSpecie();
        Hide();

        void AddSpecie()
        {
            var specie = Instantiate(plantSpecieButtonPrefab,
speciesParent).GetComponent<PlantSpecieButton>();

            specie.SetProperies(CreatePropertyList());
            specie.SetText($"Specie {specieButtons.Count}");
            specie.Pressed += () => { SetSelectedSpecie(specie); };

            specieButtons.Add(specie);
```

```
        SetSelectedSpecie(specie);
        SpecieAdded?.Invoke(specie);
        AddPressed?.Invoke();
    }

    void SetSelectedSpecie(PlantSpecieButton specie)
    {
        if (selectedSpecie == specie)
            return;

        if (selectedSpecie != null)
            SetPropertiesFormSliders(selectedSpecie.PropertyList);

        selectedSpecie = specie;
        specieNameInputField.text = selectedSpecie.SpecieName;

        specieNameInputField.text = specie.PropertyList.speciesName;
        spawnInterval.value = specie.PropertyList.spawnInterval;
        spawnPerInterval.value = specie.PropertyList.spawnsPerInterval;
        initialCount.value = specie.PropertyList.initialCount;

        SelectedSpecieChanged?.Invoke();
    }

    PlantPropertyList CreatePropertyList()
    {
        var properties = new PlantPropertyList();

        SetPropertiesFormSliders(properties);

        return properties;
    }

    void SetPropertiesFormSliders(PlantPropertyList properties)
    {
        properties.speciesId = Indexer.GetIndex();
        properties.speciesName = specieNameInputField.text;
        properties.spawnInterval = spawnInterval.value;
```

```
                properties.spawnsPerInterval = spawnPerInterval.value;
                properties.initialCount = initialCount.value;
            }
        }
    }
}
```

PopulationSettings.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using UnityEngine.UI;

namespace NaturalSelection.UI
{
    public class PopulationSettings : MonoBehaviour
    {
        [SerializeField] TMP_InputField maleInputField;
        [SerializeField] TMP_InputField femaleInputField;
        [SerializeField] Slider dispersionSlider;

        public float Dispersion => dispersionSlider.value;
        public int MaleCount => int.Parse(maleInputField.text);
        public int FemaleCount => int.Parse(femaleInputField.text);
    }
}
```

PreviewUI.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
```

```
namespace NaturalSelection.UI
{
    public class PreviewUI : UIWindow
    {
        public event Action ContinueButtonPressed;
        [SerializeField] Button continueButton;

        void Start()
        {
            Show();
            continueButton.onClick.AddListener(() =>
            {
                ContinueButtonPressed?.Invoke();
                Hide();
            });
        }
    }
}
```

Runtime.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using UnityEngine.UI;

namespace NaturalSelection.UI
{
    public class RuntimeUI : UIWindow
    {
        public event Action QuitPressed;
        public event Action<float> TimeScaleChanged;

        [SerializeField] Slider timeSlider;
        [SerializeField] Button quitButton;
```

```
        [SerializeField] Transform specieInfoListParent;
        [SerializeField] GameObject specieInfoPrefab;

        void Start()
        {
            var populationUI = UIManager.Get<SpeciePopulationUI>();

            quitButton.onClick.AddListener(() => QuitPressed?.Invoke());

            populationUI.ContinuePressed += () =>
            {
                for (int i = 0; i < populationUI.PopulationSettings.Count; i++)
                {
                    var populationSetting = populationUI.PopulationSettings[i];
                    var newInfo = Instantiate(specieInfoPrefab,
specieInfoListParent).GetComponent<SpecieInfoRuntime>();

                    newInfo
                        .SetMaleCount(populationSetting.MaleCount)
                        .SetFemaleCount(populationSetting.FemaleCount)
                        .SetName(populationUI.SpecieButtons[i].SpecieName);
                }

                Show();
            };

            timeSlider.onValueChanged.AddListener((value) =>
TimeScaleChanged?.Invoke(value));

            Hide();
        }
    }
}
```

SliderWithText.cs

```
using System.Collections;
```

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class SliderWithText : MonoBehaviour
{
    [SerializeField] Slider slider;
    [SerializeField] TextMeshProUGUI text;

    void Awake()
    {
        text.text = $"{slider.value}";
        slider.onValueChanged.AddListener((value) => { text.text = $"{value}"; });
    }
}
```

SpecieButton.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

namespace NaturalSelection.UI
{
    public class SpecieButton : MonoBehaviour
    {
        public event Action Pressed;
        public event Action<string> NameChanged;

        public string SpecieName => specieName.text;
```

```
        public AnimalPropertyList PropertyList { get => propertyList; set =>
propertyList = value; }

        [SerializeField] Button button;
        [SerializeField] TextMeshProUGUI specieName;
        AnimalPropertyList propertyList;


        public void SetProperies(AnimalPropertyList properties)
        {
            propertyList = properties;
        }

        void Start()
        {
            button.onClick.AddListener(() => Pressed?.Invoke());
        }

        public void SetText(string text)
        {
            specieName.text = text;
            NameChanged?.Invoke(text);
        }
    }
}
```

SpecieInfoRuntime.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

namespace NaturalSelection.UI
{
    public class SpecieInfoRuntime : MonoBehaviour
    {
        [SerializeField] TextMeshProUGUI specieName;
```

```
        [SerializeField] TextMeshProUGUI maleCount;
        [SerializeField] TextMeshProUGUI femaleCount;

        public SpecieInfoRuntime SetName(string name)
        {
            specieName.text = name;
            return this;
        }

        public SpecieInfoRuntime SetMaleCount(int count)
        {
            maleCount.text = $"{count}M";
            return this;
        }

        public SpecieInfoRuntime SetFemaleCount(int count)
        {
            femaleCount.text = $"{count}F";
            return this;
        }
    }
}
```

SpeciePopulationUI.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

namespace NaturalSelection.UI
{
    public class SpeciePopulationUI : UIWindow
    {
        public event Action ContinuePressed;
```

```
        [SerializeField] Transform specieButtonsParent;
        [SerializeField] Transform populationSettingsParent;
        [SerializeField] GameObject specieButtonPrefab;
        [SerializeField] GameObject speciePopulationPrefab;
        [SerializeField] Button continueButton;

        List<SpecieButton> specieButtons = new List<SpecieButton>();
        List<PopulationSettings> populationSettings = new
List<PopulationSettings>();
        public List<SpecieButton> SpecieButtons => specieButtons;
        public List<PopulationSettings> PopulationSettings =>
populationSettings;

        void Start()
        {
            var specieUI = UIManager.Get<SpecieUI>();

            specieUI.ContinuePressed += () =>
            {
                specieUI.SpecieButtons.ForEach(x =>
                {
                    var newButton = Instantiate(x,
specieButtonsParent).GetComponent<SpecieButton>();
                    newButton.SetProperies(x.PropertyList);
                    specieButtons.Add(newButton);

                    var newPopulationSetting = Instantiate(speciePopulationPrefab,
populationSettingsParent).GetComponent<PopulationSettings>();
                    populationSettings.Add(newPopulationSetting);
                });

                Show();
            };

            continueButton.onClick.AddListener(() =>
            {
                ContinuePressed?.Invoke();
                Hide();
```

```
            });

            Hide();
        }
    }
}
```

SpecieToEat.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

namespace NaturalSelection.UI
{
    public class SpecieToEat : MonoBehaviour
    {
        [SerializeField] Toggle toggle;
        [SerializeField] TextMeshProUGUI specieName;

        public bool IsSelected => toggle.isOn;
        public string SpecieName => specieName.text;

        void Start()
        {
            toggle.isOn = true;
        }

        public void SetText(string text)
        {
            specieName.text = text;
        }

        public void SubscribeTo(SpecieButton specieButton)
        {
```

```
            SetText(specieButton.SpecieName);
            specieButton.NameChanged += (newName) => SetText(newName);
        }

        public void SubscribeTo(PlantSpecieButton specieButton)
        {
            SetText(specieButton.SpecieName);
            specieButton.NameChanged += (newName) => SetText(newName);
        }
    }
}
```

SpecieUI. Cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using NaughtyAttributes;
using System;
using NaturalSelectionTool;

namespace NaturalSelection.UI
{
    public class SpecieUI : UIWindow
    {
        public event Action SelectedSpecieChanged;
        public event Action<SpecieButton> SpecieAdded;
        public event Action<SpecieButton> SpecieRemoved;
        public event Action CarniToggleChanged;
        public event Action ContinuePressed;

        [SerializeField, BoxGroup("Species")] GameObject
specieButtonPrefab;
        [SerializeField, BoxGroup("Species")] Transform speceieListParent;
        [SerializeField, BoxGroup("Species")] Button addButton;
```

```
        [SerializeField, BoxGroup("Species")] Button removeButton;
        [SerializeField, BoxGroup("Species")] TMP_InputField
specieNameInputField;

        [SerializeField, BoxGroup("Toggle")] Toggle herbiToggle;
        [SerializeField, BoxGroup("Toggle")] Toggle carniToggle;

        [SerializeField, BoxGroup("Species to eat")] Transform
speceieToEatListParent;
        [SerializeField, BoxGroup("Species to eat")] GameObject
speceieToEatPrefab;
        [SerializeField, BoxGroup("Species to eat")] GameObject
plantSpeceieToEatPrefab;

        [SerializeField, BoxGroup("Slider")] Slider speed;
        [SerializeField, BoxGroup("Slider")] Slider senseRadius;
        [SerializeField, BoxGroup("Slider")] Slider startMatingAge;
        [SerializeField, BoxGroup("Slider")] Slider endMatingAge;
        [SerializeField, BoxGroup("Slider")] Slider thirstTrigger;
        [SerializeField, BoxGroup("Slider")] Slider hungerTrigger;
        [SerializeField, BoxGroup("Slider")] Slider hungerPerSecond;
        [SerializeField, BoxGroup("Slider")] Slider thirstPerSecond;
        [SerializeField, BoxGroup("Slider")] Slider pregnantTime;
        [SerializeField, BoxGroup("Slider")] Slider offspringsCount;
        [SerializeField, BoxGroup("Slider")] Slider malePartInOfsprings;
        [SerializeField, BoxGroup("Slider")] Slider offspringDevelopment;
//"Развитость" потомства, какой процент от своих будущих статов они
получают
        [SerializeField, BoxGroup("Slider")] Slider mutationChance;
        [SerializeField, BoxGroup("Slider")] Slider mutationStrong;
        [SerializeField, BoxGroup("Slider")] Slider maxMutationStrong;

        [SerializeField] Button continueButton;

        List<SpecieButton> specieButtons = new List<SpecieButton>();
        List<SpecieToEat> speciesToEat = new List<SpecieToEat>();
        List<SpecieToEat> plantSpeciesToEat = new List<SpecieToEat>();
        SpecieButton selectedSpecie;
```

```
        List<AnimalPropertyList> createdProperties = new
List<AnimalPropertyList>();

        public List<SpecieButton> SpecieButtons { get => specieButtons; set =>
specieButtons = value; }

        void Start()
        {
            HandleSpecieList();
            HandleToggles();
            HandleSpeciesToEat();
            HandleSliders();

            AddSpecie();
            Hide();

            UIManager.Get<PlantsUI>().ContinueButtonPressed += () =>
Show();

            void AddSpecie()
            {
                var specie = Instantiate(specieButtonPrefab,
speceieListParent).GetComponent<SpecieButton>();

                specie.SetProperies(CreatePropertyList());
                specie.SetText($"Specie {specieButtons.Count}");
                specie.Pressed += () => { SetSelectedSpecie(specie); };

                SetSelectedSpecie(specie);
                specieButtons.Add(specie);
                SpecieAdded?.Invoke(specie);
            }

            void HandleSliders()
            {
                speed.value = StaticRandom.Instance.Next(30, 70);
                senseRadius.value = StaticRandom.Instance.Next(30, 70);
                startMatingAge.value = StaticRandom.Instance.Next(30, 70);
```

```
            endMatingAge.value = StaticRandom.Instance.Next(30, 70);
            thirstTrigger.value = StaticRandom.Instance.Next(30, 70);
            hungerTrigger.value = StaticRandom.Instance.Next(30, 70);
            hungerPerSecond.value = StaticRandom.Instance.Next(30, 70);
            thirstPerSecond.value = StaticRandom.Instance.Next(30, 70);
            pregnantTime.value = StaticRandom.Instance.Next(30, 70);
            offspringsCount.value = StaticRandom.Instance.Next(30, 70);
            malePartInOfsprings.value = StaticRandom.Instance.Next(30, 70);
            offspringDevelopment.value = StaticRandom.Instance.Next(30,
70);
            mutationChance.value = StaticRandom.Instance.Next(30, 70);
            mutationStrong.value = StaticRandom.Instance.Next(30, 70);
            // maxMutationStrong.value = StaticRandom.Instance.Next(30,
70);
        }

        continueButton.onClick.AddListener(() =>
        {
            SetPropertiesFormSliders(selectedSpecie.PropertyList);

            Debug.Log(selectedSpecie.PropertyList.speed);
            ContinuePressed?.Invoke();
            Hide();
        });

        void HandleToggles()
        {
            herbiToggle.onValueChanged.AddListener((active) =>
            {
                carniToggle.isOn = !active;
                CarniToggleChanged?.Invoke();
            });

            carniToggle.onValueChanged.AddListener((active)          =>
herbiToggle.isOn = !active);

            herbiToggle.isOn = true;
            carniToggle.isOn = false;
```

```
            SelectedSpecieChanged += () =>
            {
                // change toggle according to specie
            };
        }

        void HandleSpeciesToEat()
        {

speceieToEatListParent.GetComponentsInChildren<SpecieToEat>(speciesToEat);

            for (int i = speciesToEat.Count; i < specieButtons.Count; i++)
                AddAnimalSpecieToEat();

            for (int i = 0; i < specieButtons.Count; i++)
            {
                // neobhodimo chtobi rabotalo vinesti v peremennie
                var specieButton = specieButtons[i];
                var specieToEat = speciesToEat[i];

                specieToEat.SubscribeTo(specieButton);
            }


            UIManager.Get<PlantsUI>().ContinueButtonPressed += () =>
            {
                UIManager.Get<PlantsUI>().SpecieButtons.ForEach(x =>
                {
                    AddPlantSpecieToEat(x);
                });
            };

            CarniToggleChanged += () =>
{ SetSpeciesToEatShow(carniToggle.isOn); };
            herbiToggle.onValueChanged.AddListener((value) =>
plantSpeciesToEat.ForEach(x =>
            {
```

```
            x.gameObject.SetActive(value);
        }));

        SelectedSpecieChanged += () => { SetSpeciesToEatShow(carniToggle.isOn); };

        SpecieAdded += (specie) =>
        {
            AddAnimalSpecieToEat(specie);
            SetSpeciesToEatShow(carniToggle.isOn);
        };

        SpecieRemoved += (specie) =>
        {
            var removedSpecie = speciesToEat.Find(x => x.SpecieName == specie.SpecieName);
            Destroy(removedSpecie.gameObject);
            speciesToEat.Remove(removedSpecie);
        };


        void SetSpeciesToEatShow(bool show) { speciesToEat.ForEach(x => x.gameObject.SetActive(show)); }


        void AddAnimalSpecieToEat(SpecieButton specie = null)
        {
            var newSpecieToEat = Instantiate(speceieToEatPrefab, speceieToEatListParent).GetComponent<SpecieToEat>();

            if (specie != null)
                newSpecieToEat.SubscribeTo(specie);

            speciesToEat.Add(newSpecieToEat);
        }

        void AddPlantSpecieToEat(PlantSpecieButton specie)
        {
```

```
                var newSpecieToEat = Instantiate(plantSpeceieToEatPrefab,
speceieToEatListParent).GetComponent<SpecieToEat>();
                newSpecieToEat.SubscribeTo(specie);

                plantSpeciesToEat.Add(newSpecieToEat);
            }
        }

        void HandleSpecieList()
        {

speceieListParent.GetComponentsInChildren<SpecieButton>(specieButtons);

            if (specieButtons.Count > 0)
            {
                specieButtons.ForEach(x =>
                {
                    x.Pressed += () => { SetSelectedSpecie(x); };
                    x.SetProperies(CreatePropertyList());
                });

                SetSelectedSpecie(specieButtons[0]);
            }

            addButton.onClick.AddListener(() =>
            {
                AddSpecie();
            });

            removeButton.onClick.AddListener(() =>
            {
                if (selectedSpecie == null)
                    return;

                SpecieRemoved?.Invoke(selectedSpecie);
                Destroy(selectedSpecie.gameObject);
                specieButtons.Remove(selectedSpecie);
```

```
            if (specieButtons.Count > 0)
                SetSelectedSpecie(specieButtons[specieButtons.Count - 1]);
        });

        specieNameInputField.onValueChanged.AddListener((value) =>
        {
            selectedSpecie.SetText(value);
        });
    }

    AnimalPropertyList CreatePropertyList()
    {
        var properties = new AnimalPropertyList();

        SetPropertiesFormSliders(properties);

        return properties;
    }

    void SetSelectedSpecie(SpecieButton specie)
    {
        if (selectedSpecie == specie)
            return;

        if (selectedSpecie != null)
            SetPropertiesFormSliders(selectedSpecie.PropertyList);
        selectedSpecie = specie;
        specieNameInputField.text = selectedSpecie.SpecieName;

        specieNameInputField.text = specie.PropertyList.speciesName;
        speed.value = specie.PropertyList.speed;
        thirstTrigger.value = specie.PropertyList.thirstTrigger;
        thirstPerSecond.value                                          =
specie.PropertyList.thirstDecreasePerSecond;
        hungerPerSecond.value                                          =
specie.PropertyList.hungerDecreasePerSecond;
        hungerTrigger.value = specie.PropertyList.hungerTrigger;
        startMatingAge.value = specie.PropertyList.startMatingAge;
```

```
                endMatingAge.value = specie.PropertyList.endMatingAge;
                senseRadius.value = specie.PropertyList.senseRadius;
                pregnantTime.value = specie.PropertyList.pregnantTime;
                offspringsCount.value = specie.PropertyList.offspringsCount;
                offspringDevelopment.value                                    =
specie.PropertyList.offspringDevelopment;
                mutationChance.value = specie.PropertyList.mutationChance;
                mutationStrong.value = specie.PropertyList.mutationStrong;

                SelectedSpecieChanged?.Invoke();
            }

            void SetPropertiesFormSliders(AnimalPropertyList properties)
            {
                properties.speciesId = Indexer.GetIndex();
                properties.speciesName = specieNameInputField.text;
                properties.speed = (int)speed.value;
                properties.thirstTrigger = thirstTrigger.value;
                properties.thirstDecreasePerSecond = thirstPerSecond.value;
                properties.hungerDecreasePerSecond = hungerPerSecond.value;
                properties.hungerTrigger = hungerTrigger.value;
                properties.startMatingAge = startMatingAge.value;
                properties.endMatingAge = endMatingAge.value;
                properties.senseRadius = senseRadius.value;
                properties.pregnantTime = pregnantTime.value;
                properties.offspringsCount = (int)offspringsCount.value;
                properties.offspringDevelopment = offspringDevelopment.value;
                properties.mutationChance = mutationChance.value;
                properties.mutationStrong = mutationStrong.value;
            }
        }
    }
}
```

UIWindow.cs

```
using System;
```

```
using System.Collections;
using System.Collections.Generic;

using UnityEngine;

namespace NaturalSelection.UI
{
   public class UIWindow : MonoBehaviour
   {
      protected GameObject content;

      protected virtual void Awake()
      {
         gameObject.SetActive(true);
         content = transform.GetChild(0).gameObject;
      }

      protected virtual void Show()
      {
         content.SetActive(true);
      }

      protected virtual void Hide()
      {
         content.SetActive(false);
      }
   }
}
```

ExtentionMethods.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Threading;
using NaturalSelection;
using UnityEngine;
```

```
namespace NaturalSelectionTool
{
   public static class ExtentionMethods
   {
      public static float GetDistanceTo(this Vector3 pos1, Vector3 pos2)
      {
         Vector3 heading;

         heading.x = pos1.x - pos2.x;
         heading.y = pos1.y - pos2.y;
         heading.z = pos1.z - pos2.z;

         var distanceSquared = heading.x * heading.x + heading.y * heading.y
+ heading.z * heading.z;

         return Mathf.Sqrt(distanceSquared);
      }

      public static float GetDistanceTo(this Transform t1, Transform t2)
      {
         return t1.position.GetDistanceTo(t2.position);
      }


      public static float GetDistanceTo(this Transform t, Vector3 v)
      {
         return t.position.GetDistanceTo(v);
      }

      public static float GetDistanceTo(this Vector3 v, Transform t)
      {
         return t.position.GetDistanceTo(v);
      }

      public static float GetDistanceTo(this GameObject go1, GameObject
go2)
      {
```

```
            return
go1.transform.position.GetDistanceTo(go2.transform.position);
        }

        public static float GetDistanceTo(this GameObject go1, Transform t)
        {
            return go1.transform.position.GetDistanceTo(t.position);
        }

        public static T GetRandomItem<T>(this List<T> list)
        {
            if (list == null || list.Count == 0)
            {
                Debug.LogWarning("can't get random item: list is null or count ==
0");
                return default(T);
            }

            //          UnityEngine.Random.InitState(UnityEngine.Random.Range(-
10000, 10000));
            return list[StaticRandom.Instance.Next(0, list.Count)];
        }

        public static T GetRandomItem<T>(this List<T> list, out int index)
        {
            if (list == null || list.Count == 0)
            {
                Debug.LogWarning("can't get random item: list is null or count ==
0");
                index = 0;
                return default(T);
            }

            index = StaticRandom.Instance.Next(0, list.Count);
            return list[index];
        }

        public static int GetRandomIndex<T>(this List<T> list)
```

```
        {
            if (list == null || list.Count == 0)
            {
                Debug.LogWarning("can't get random item: list is null or count ==
0");
                return -1;
            }

            return StaticRandom.Instance.Next(0, list.Count);
        }

        public static int GetNextIndex<T>(this List<T> list, int curIndex)
        {
            if (list == null || list.Count == 0)
            {
                Debug.LogWarning("Can't get next index: list is null or count is 0");
                return -1;
            }

            if (list.Count == 1)
            {
                return 0;
            }

            var nextIndex = curIndex + 1;
            return nextIndex > list.Count - 1 ? 0 : nextIndex;
        }



        public static int GetPreviousIndex<T>(this List<T> list, int curIndex)
        {
            if (list == null || list.Count == 0)
            {
                Debug.LogWarning("Can't get previous index: list is null or count
is 0");
                return -1;
            }
```

```
            var nextIndex = curIndex - 1;
            return nextIndex < 0 ? list.Count - 1 : nextIndex;
        }

        public static T GetRandomItem<T>(this T[] array)
        {
            if (array == null || array.Length == 0)
            {
                Debug.LogWarning("can't get random item: list is null or count ==
0");
                return default(T);
            }

            return array[UnityEngine.Random.Range(0, array.Length)];
        }

        public static bool IsNullOrEmpty<T>(this List<T> list) => list == null ||
list.Count == 0;

        public static T GetRandomEnum<T>()
        {
            var v = Enum.GetValues(typeof(T));
            return (T)v.GetValue(new System.Random().Next(v.Length));
        }

        public static T GetRandomEnum<T>(int excludeNum)
        {
            var v = Enum.GetValues(typeof(T));
            return (T)v.GetValue(new System.Random().Next(v.Length -
excludeNum));
        }



        public static List<GameObject> FindGameObjectsInChildWithTag(this
GameObject parent, string tag)
        {
```

```
            var t = parent.transform;
            var list = new List<GameObject>(8);

            for (int i = 0; i < t.childCount; i++)
            {
                var child = t.GetChild(i);

                if (child.CompareTag(tag))
                {
                    list.Add(child.gameObject);
                }

            }

            return list;
        }


        ///<summary>
        /// Return percernt of value
        ///</summary>
        public static float GetPercent(this float value, float desiredPercent) =>
            value / 100f * desiredPercent;

        public static float GetPercent2(this float value, float desiredPercent) =>
            value * 100f / desiredPercent;

        ///<summary>
        /// Return percernt of value
        ///</summary>
        public static float GetPercent(this int value, int desiredPercent) =>
            (float)value / 100f * (float)desiredPercent;

        public static int Replace<T>(this IList<T> source, T oldValue, T
newValue)
        {
            if (source == null)
                throw new ArgumentNullException("source");
```

```
        var index = source.IndexOf(oldValue);
        if (index != -1)
            source[index] = newValue;
        return index;
    }


    public static Vector3 GetBezierPoint(Vector3 startP, Vector3 midP,
Vector3 endP, float t)
    {
        return Vector3.Lerp(
            Vector3.Lerp(startP, midP, t),
            Vector3.Lerp(midP, endP, t),
            t
        );
    }


    ///<summary>
    /// Return random element from given probabilites
    ///</summary>
    public static int RollDice(this List<int> probabilities)
    {
        var total = 0f;

        probabilities.ForEach(probability => total += probability);

        var randomProbability = StaticRandom.Instance.Next(0, (int)total);

        for (int i = 0; i < probabilities.Count; i++)
            if (randomProbability < probabilities[i])
                return i;
            else
                randomProbability -= probabilities[i];
        return -1;
    }
```

```
        public static int RollDice(this List<float> probabilities)
        {
            var total = 0f;

            probabilities.ForEach(probability => total += probability);

            var randomProbability = (float)StaticRandom.Instance.Next(0,
(int)total);

            for (int i = 0; i < probabilities.Count; i++)
                if (randomProbability < probabilities[i])
                    return i;
                else
                    randomProbability -= probabilities[i];
            return -1;
        }

        public static GameObject CreateTestCube(Color color)
        {
            var cube = GameObject.CreatePrimitive(PrimitiveType.Cube);
            UnityEngine.Object.Destroy(cube.GetComponent<BoxCollider>());

            cube.GetComponent<MeshRenderer>().material.color = color;
            return cube;
        }

        public static GameObject CreateTestCube(Color color, Vector3 scale)
        {
            var cube = CreateTestCube(color);

            cube.transform.localScale = scale;

            return cube;
        }

        public static List<GameObject> GetAllChilds(this GameObject go)
        {
            var transform = go.transform;
```

```
            var list = new List<GameObject>(transform.childCount);

            for (int i = 0; i < transform.childCount; i++)
            {
                list.Add(transform.GetChild(i).gameObject);
            }

            return list;
        }
    }

    public static class IndexGiver
    {
        static int index;

        public static int GetIndex()
        {
            var prevIndex = index;
            index++;

            return prevIndex;
        }
    }

    public static class Get
    {
        static Dictionary<float, WaitForSeconds> delays = new
Dictionary<float, WaitForSeconds>(100);
        public static WaitForEndOfFrame EndOfFrame { get; } = new
WaitForEndOfFrame();
        public static WaitForFixedUpdate FixedUpdate { get; } = new
WaitForFixedUpdate();

        public static WaitForSeconds DelayInSeconds(float delay)
        {
            if (delays.TryGetValue(delay, out var value))
                return value;
```

```
            var newDelay = new WaitForSeconds(delay);
            delays.Add(delay, newDelay);

            return newDelay;
        }
    }

    public static class Do
    {
        public static void After(float delayInSeconds, Action action)
        {
            AppManager.Instance.StartCoroutine(Wait());
            IEnumerator Wait()
            {
                yield return Get.DelayInSeconds(delayInSeconds);
                action.Invoke();
            }
        }
    }

    public interface IValidatable
    {
        void DoAfterValidate(Action action);
    }

    public class Validator
    {
        event Action Validated;
        Func<bool>[] conditions;

        public Validator(Func<bool>[] conditions)
        {
            this.conditions = conditions;
        }

        public Validator(Func<bool> condition)
        {
            conditions = new Func<bool>[1];
```

```
            conditions[0] = condition;
        }

        public bool Check()
        {
            if (conditions == null || conditions.Length == 0)
            {
                Validated?.Invoke();
                return true;
            }

            for (int i = 0; i < conditions.Length; i++)
                if (conditions[i]() == false)
                    return false;

            Validated?.Invoke();
            return true;
        }

        public Validator DoAfterValidate(Action action)
        {
            if (Check())
                action?.Invoke();
            else
                Validated += OnValidate;

            void OnValidate()
            {
                action?.Invoke();
                Validated -= OnValidate;
            }

            return this;
        }


    }
```

```
    public static class JsonHelper
    {
        public static T[] FromJson<T>(string json)
        {
            Wrapper<T> wrapper = JsonUtility.FromJson<Wrapper<T>>(json);
            return wrapper.Items;
        }

        public static string ToJson<T>(T[] array)
        {
            Wrapper<T> wrapper = new Wrapper<T>();
            wrapper.Items = array;
            return JsonUtility.ToJson(wrapper);
        }

        public static string ToJson<T>(T[] array, bool prettyPrint)
        {
            Wrapper<T> wrapper = new Wrapper<T>();
            wrapper.Items = array;
            return JsonUtility.ToJson(wrapper, prettyPrint);
        }

        [Serializable]
        private class Wrapper<T>
        {
            public T[] Items;
        }
    }
}
```

StaticRandom.cs

```
using System;
using System.Threading;

namespace NaturalSelectionTool
```

```
    {
        public static class StaticRandom
        {
            private static int seed;

            private static ThreadLocal<System.Random> threadLocal = new
ThreadLocal<System.Random>
                    (() => new System.Random(Interlocked.Increment(ref seed)));

            static StaticRandom() => seed = Environment.TickCount;

            public static System.Random Instance { get { return
threadLocal.Value; } }
        }
    }
```

Singleton.cs

```
using System.Collections;
    using System.Collections.Generic;
    using UnityEngine;

    namespace NaturalSelection
    {
        public class Singleton<T> : MonoBehaviour where T : MonoBehaviour
        {
            static T instance;

            public static T Instance
            {
                get
                {
                    if (instance == null)
                    {
                        var instanceInScene = GameObject.FindObjectOfType<T>();
                        instance = instanceInScene;
                        return instance;
```

```
                }

                return instance;
            }
            set
            {
                if (instance == null)
                {
                    instance = value;
                }
                else
                {
                    Destroy(value.gameObject);
                    Debug.LogWarning($"Deleting other {typeof(T).Name}
singleton instance");
                }
            }
        }

        protected virtual void Awake()
        {
            if (Instance == null)
            {
                Instance = this as T;
            }
        }
    }
}
```